SIMULATION DE FLUIDES, APPROCHE LAGRANGIENNE

par

Adrien Wattez

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 20 novembre 2014

Le 20 novembre 2014

Le jury a accepté le mémoire de Adrien Wattez dans sa version finale

Membres du jury

Professeur Richard Egli
Directeur
Département d'informatique de l'Université de Sherbrooke

Professeur Fabrice Colin
Codirecteur
Département de mathématiques et d'informatique de l'Université Laurentienne

Professeur Jean-Pierre Dussault
Codirecteur
Département d'informatique de l'Université de Sherbrooke

Professeur Marie-Flavie Auclair-Fortier
Président-rapporteur
Département d'informatique de l'Université de Sherbrooke

Professeur Djemel Ziou
Membre interne
Département d'informatique de l'Université de Sherbrooke

Sommaire

Avec la généralisation du recours à l'infographie dans l'industrie des loisirs, la demande concernant la production de scènes de simulation de fluides d'un réalisme croissant a fortement augmenté durant les deux dernières décennies. Nous proposons de nombreux éléments pertinents pour simuler le fluide, essentiellement tournés vers l'approche lagrangienne (les méthodes particulaires). Ce mémoire a donc pour objet l'étude et la mise au point de techniques permettant de reproduire le comportement des fluides s'appuyant sur l'aspect particulaire du fluide. Les algorithmes de ces dernières années permettent un gain de performance significatif, nous permettant d'obtenir des simulations de fluides incompressibles en temps réel. L'usage des noyaux constants par morceaux, nouvel outil de calcul numérique, au sein de simulations de fluides dites lagrangiennes sera également abordé. Avec l'augmentation continue de la puissance de calcul et de nouvelles avancées telles que la programmation dite GPGPU, nous verrons également comment obtenir une recherche de voisinage efficace permettant d'augmenter grandement les performances de calcul.

Mots-clés: Simulation de fluides ; Méthode lagrangienne ; SPH ; Équations de Navier-Stokes ; Noyaux constants par morceaux ; GPGPU ; CUDA.

Remerciements

Je tiens d'abord à remercier mon directeur de recherche Monsieur Richard Egli, professeur à l'université de Sherbrooke, pour sa confiance et pour m'avoir donné l'opportunité d'approfondir ce sujet.

Je remercie également mes codirecteurs, Monsieur Jean-Pierre Dussault, professeur à l'université de Sherbrooke, Monsieur Fabrice Colin, professeur à l'université Laurentienne, pour l'aide indispensable qui m'a permis d'avancer dans mon projet et leurs idées toujours pertinentes.

Je remercie les membres du jury, Madame Marie-Flavie Auclair-Fortier, pour avoir accepté de présider ce jury et Monsieur Djemel Ziou, professeur à l'université de Sherbrooke.

Je remercie aussi tous mes amis rencontrés au Québec, qui m'ont permis de découvrir d'autres horizons et d'autres cultures.

Je remercie mes collègues de laboratoire, le BISOUS, le DOMUS, le SMART et le MOIVRE pour leurs conseils avisés et leur accueil chaleureux.

Je remercie Charlotte, ma conjointe, pour son soutien et son accompagnement dans ma vie (au Québec et dans le monde entier).

Abréviations

AOS Array Of Structure (Tableau de structures)

API Application Programming Interface (Interface de programmation d'application)

AV Artificial Viscosity (Viscosité artificielle)

BLAS Basic Linear Algebra Subprograms
(Sous-programme d'algèbre linéaire de base)

CFL Courant–Friedrichs–Lewy (Critère de stabilité)

CPU Central Processing Unit (Unité centrale de traitement)

CSTPM Constant par morceaux

CUDA Compute Unified Device Architecture
(Architecture de dispositif unifiée pour le calcul)

DEM Discrete Element Method (Méthode des éléments discrets)

DRAM Dynamic Random-Access Memory (Mémoire vive dynamique)

FLIP Fluid-Implicit-Particle (Particules de fluide implicite)

GLPK GNU Linear Programming Kit (Kit de programmation linéaire GNU)

GLSL OpenGL Shading Language (langage OpenGL de shaders)

GPGPU General-Purpose computation on Graphic Processing Units
(Calcul universel sur unités de traitement graphique)

GPU Graphics Processing Unit (Unité de traitement graphique)

HPC High-Performance Computing (Calcul à rendement élevé)

IPS Image par seconde

LINPRO LINear PROgramming solver (Solveur de programmation linéaire)

Abréviations

LP Linear Programming (Programmation linéaire)

NVCC Nvidia CUDA compiler (Compilateur de langage CUDA Nvidia)

OpenAcc Open Accelerators (Bibliothèque libre de langage pour accélérateurs)

OpenCL Open Computing Language (Langage ouvert de calculs parallèles)

OpenGL Open Graphics Library (Bibliothèque libre de langage graphique)

PBD Position Based Dynamics (Dynamique basée sur la position)

PBF Position Based Fluids (Simulation de fluide basée sur la position)

PCISPH Predictive-Corrective Incompressible SPH (Simulation SPH incompressible, prédite et corrective)

PIC Particle-In-Cell (Particules dans des cases)

QUAPRO linear QUAdratic PROgramming solver (Solveur de programmation linéaire quadratique)

SDK Software Development Kit (Kit de développement de logiciel)

SM-SMX Streaming Multiprocessor (Multiprocesseur de flux)

SOA Structure Of Array (Structure de tableaux)

SPH Smoothed-Particle Hydrodynamics (Particules hydrodynamiques lissées)

SRAM Static Random-Access Memory (Mémoire vive statique)

WCSPH Weakly Compressible SPH (Simulation SPH faiblement compressible)

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

Introduction

Un fluide est un corps susceptible de s'écouler, se déformer facilement. Il est considéré comme un milieu matériel continue. On discerne les fluides compressibles élastiques telles que les gaz mais également les fluides peu compressibles (incompressibles) apparentés aux liquides. On observe depuis ces dernières années un intérêt continu dans le domaine de la simulation, de la reproduction de comportements complexes ou de visuels de fluides. La demande et les attentes concernant la production de scènes d'un réalisme quasi-parfait ont fortement augmenté durant les deux dernières décennies. Par ailleurs les méthodes empiriques basées sur des interventions manuelles ne suffisent plus à donner satisfaction à grande échelle ou dans des situations complexes. La simulation de fluides est présente dans deux principales industries : le cinéma et l'industrie du jeux vidéo, ou encore l'infographie. Le dernier film «La Reine des neiges» de Disney («Frozen» en anglais), en témoigne avec les dernières avancées de la simulation de neige [SSC+13]. Dans ce mémoire, de nombreux éléments pertinents pour les simulations de fluides seront abordés, essentiellement tournés vers les méthodes particulaires. Ces méthodes basées sur l'approche lagrangienne deviennent une alternative de plus en plus attrayante par rapport à l'approche eulérienne qui repose sur un maillage traditionnel et des concepts basés sur des grilles. Ce mémoire a donc pour objet l'étude et la mise au point de techniques permettant de reproduire le comportement des fluides facilitant l'inclusion des liquides dans des scènes réalisées en images de synthèse.

Au sein du chapitre 1, nous présentons la dynamique des fluides et l'ensemble des équations qui permettent de décrire le comportement du fluide en mouvement. Historiquement, les premières équations de conservation de la masse et de mouvement du fluide ont été obtenues par Leonhard Euler avant Henri Navier et George Gabriel

Stokes. Utilisant une approche physique, ce chapitre décrit l'obtention de ces équations ; l'annexe A permettant de mieux comprendre la notation utilisée. Ce chapitre décrit également la complexité mathématique de ces équations et les approches (eulériennes/lagrangiennes) à adopter afin de les résoudre. Bien souvent, en infographie, les équations qui régissent le mouvement des fluides sont simplifiées afin de conserver d'excellentes performances de calcul mais aussi des rendus réalistes.

Par la suite, dans le chapitre 2, divers principes mathématiques des méthodes particulaires sont présentés. Notamment la méthode SPH («Smoothed Particle Hydrodynamics» en anglais) qui est souvent présentée comme une méthode particulaire pure. Introduite par Lucy [Luc77] en 1977, ainsi que Gingold et Monaghan [GM77], elle est d'abord appliquée à la modélisation d'amas stellaires en astrophysique, puis très vite adoptée en infographie [MCG03] dans le but d'être utilisée dans l'industrie de l'animation. Nous retrouvons également dans le chapitre 2 une discussion des noyaux applicables aux méthodes particulaires.

À l'aide des outils mathématiques définis dans le chapitre 2, de nombreux systèmes d'équations aux dérivées partielles ou ordinaires peuvent être résolus. Le but de ce mémoire est d'appliquer les méthode particulaires à l'étude des simulations de fluides. Nous ne passerons pas en revue tous les modèles particulaires qui ont pu être utilisés dans les nombreux domaines de la physique. Nous allons seulement décrire quelques stratégies clefs de ces dernières années dans le domaine de l'infographie à l'aide de la méthode SPH. Pour cela, le chapitre 3 permet de comprendre l'utilisation des principes mathématiques des méthodes particulaires dans un contexte de simulation de fluides, afin de résoudre les équations de Euler/Navier-Stokes.

Le chapitre 4 complète le chapitre 3. En approche lagrangienne, le fluide est traditionnellement comme en physique considéré comme faiblement compressible. La raison est qu'il est nettement plus facile de calculer la pression à partir d'une équation d'état décrite dans le chapitre 3, plutôt que d'avoir à obtenir cette pression par le biais de la résolution d'une équation (à l'image des méthodes eulériennes qui utilisent une équation de Poisson). Le chapitre 4 présente les algorithmes et corrections adoptés afin de rendre le fluide compressible, incompressible. Les algorithmes performants de ces dernières années offrent un gain de performance significatif, nous permettant d'obtenir des simulations de fluides de qualité en temps réel.

Dans la littérature, plusieurs noyaux sont proposés afin de permettre une bonne simulation des forces évaluées à l'aide du gradient ou du Laplacien dans le but de résoudre les équations de la dynamique des fluides. Au sein du chapitre 5, nous proposons d'apporter une utilisation des noyaux constants par morceaux, développés par Jean-Marc Belley, Philippe Belley, Fabrice Colin et Richard Egli [BBCE09]. Ces noyaux se présentent comme un compromis entre les noyaux classiques utilisés en approche lagrangienne et les différences finies eulériennes. Leur usage en une dimension dans l'article [BBCE09], permet de mettre en valeur et ce en comparaison avec les méthodes SPH classiques, une grande amélioration dans les estimations des valeurs d'une fonction et de ses deux premières dérivées. Nous appliquons ces noyaux en deux dimensions dans une approche lagrangienne. L'estimation à l'aide de ces noyaux est basée sur une résolution matricielle [CESM11]. Nous utilisons également une approche de programmation linéaire, permettant de fournir des résultats encourageant pour une éventuelle utilisation dans des méthodes eulériennes.

Malheureusement, la simulation de fluides nécessite souvent des algorithmes très complexes et un temps de calcul important pour représenter la façon dont le fluide évolue avec l'environnement. Par conséquent, ces simulations sont généralement limitées à un rendu dit hors ligne («off line» en anglais). Cependant, avec l'augmentation continue de la puissance de calcul et de nouvelles avancées tels que les GPU («Graphics Processing Unit» en anglais), nous pouvons simuler des fluides en temps réel pour des applications interactives comme les jeux vidéo. Le chapitre 6 porte sur la programmation dite hautement parallèle. Méthodes et principes de programmation sont décrits dans le but d'obtenir un programme de rendu des fluides en trois dimensions. Ce chapitre 6 se concentre essentiellement sur les pratiques de programmation dite GPGPU («General-Purpose computation on Graphic Processing Units» en anglais) mais aussi sur la recherche de voisinages qui, dans un contexte de simulations particulaires, permet d'augmenter grandement les performances de calcul.

Chapitre 1

Introduction à la dynamique et à la simulation des fluides

Décrire le mouvement d'un fluide fait appel à des notions différentes de celles développées en mécanique du point (mouvement et physique du point matériel) ou du solide. Le mouvement d'un fluide est un écoulement où il y a déformation continue du fluide. On peut isoler (par la pensée ou en trouvant un moyen de visualisation, coloration par exemple) une partie restreinte du fluide et en déduire le comportement et les contraintes que subissent le fluide. La dynamique des fluides s'attache à décrire précisément le mouvement des fluides au sein d'un écoulement, en le reliant aux différentes forces en présence. L'objectif est donc de mettre en place une équation locale qui puisse rendre compte du lien entre vitesse, pression, forces de volume et de frottement (viscosité). Avant de lire ce qui suit, il peut être judicieux de prendre connaissance du contenu de l'annexe A pour mieux appréhender la notation scientifique utilisée.

1.1 Les équations d'Euler

Dans la dynamique des fluides, les équations d'Euler sont un ensemble d'équations régissant l'écoulement du fluide non visqueux. Les équations traduisent les principes physiques suivant : la conservation de la masse (la continuité), de la quantité de

mouvement (moment) et de l'énergie, correspondant aux équations de Navier-Stokes en considérant la viscosité nulle et en l'absence d'échange de chaleur. Historiquement, les équations de conservation de la masse et du mouvement ont été obtenues par Euler avant Navier-Stokes (une description de ces équations est faite dans la section 1.2). Cet ensemble d'équations doit son nom à Leonhard Euler.

1.1.1 L'équation de conservation du mouvement d'Euler

La démonstration ci-dessous s'inspire de celle du livre [And95] et permet de démontrer d'une façon très simple de quelle manière Euler a obtenu l'équation de conservation de la quantité de mouvement («momentum equation» en anglais). Considérant le champ de vecteurs vitesse $\vec{v}$ (en $m \cdot s^{-1}$) défini en deux dimensions :

$$\vec{v} = (v_x, v_y) = (v_x(x, y, t), v_y(x, y, t)), \tag{1.1}$$

ainsi que le principe des différentielles totales utilisé par Euler :

$$dv_x = \frac{\partial v_x}{\partial x}\Delta x + \frac{\partial v_x}{\partial y}\Delta y + \frac{\partial v_x}{\partial t}\Delta t, \tag{1.2}$$

$$dv_y = \frac{\partial v_y}{\partial x}\Delta x + \frac{\partial v_y}{\partial y}\Delta y + \frac{\partial v_y}{\partial t}\Delta t, \tag{1.3}$$

ensuite, en prenant en compte le temps à l'aide de Δt, on obtient :

$$\frac{dv_x}{dt} = \frac{\partial v_x}{\partial x}\frac{\Delta x}{dt} + \frac{\partial v_x}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial v_x}{\partial t}, \tag{1.4}$$

$$\frac{dv_y}{\Delta t} = \frac{\partial v_y}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial v_y}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial v_y}{\partial t}. \tag{1.5}$$

Toutefois, il est nécessaire de se rappeler :

$$v_x = \frac{\Delta x}{\Delta t} = \frac{dx}{dt}\,;\ \text{et}\ \ v_y = \frac{\Delta y}{\Delta t} = \frac{dy}{dt}. \tag{1.6}$$

On retrouve donc, avec une simple substitution, les dérivées totales suivantes :

$$\frac{dv_x}{dt} = \frac{\partial v_x}{\partial x}v_x + \frac{\partial v_x}{\partial y}v_y + \frac{\partial v_x}{\partial t}, \tag{1.7}$$

$$\frac{dv_y}{dt} = \frac{\partial v_y}{\partial x}v_x + \frac{\partial v_y}{\partial y}v_y + \frac{\partial v_y}{\partial t}. \tag{1.8}$$

Euler s'appuie sur la deuxième loi de Newton. Les expressions précédentes représentent l'accélération d'un élément de fluide. Quant aux forces, Euler identifie au sein du fluide la pression, la friction et la gravité. Pour le moment, abandonnons la friction et mettons l'accent sur la pression et la gravité. Soit la force de gravité suivante :

$$\vec{F_g} = \vec{g} = (0, -Mg)\,, \tag{1.9}$$

où g représente la constante de gravité souvent appelée pesanteur, $g = 9.81\, m \cdot s^{-2}$. M (en kg) représente la masse totale du fluide :

$$M = \rho V. \tag{1.10}$$

V (en m^3 en trois dimensions) représente le volume du fluide et ρ la fonction de densité (en $kg \cdot m^{-3}$ en trois dimensions et $kg \cdot m^{-2}$ en deux dimensions) représente une grandeur physique qui caractérise la masse du fluide par unité de volume. En deux dimensions, on se retrouve avec une portion du fluide, c'est-à-dire une surface discrétisée ($\Delta x\ \Delta y$) qui représente un rectangle du fluide figure 1.1. La masse locale du rectangle de fluide est donc :

$$M_L = \rho\ \Delta x \Delta y. \tag{1.11}$$

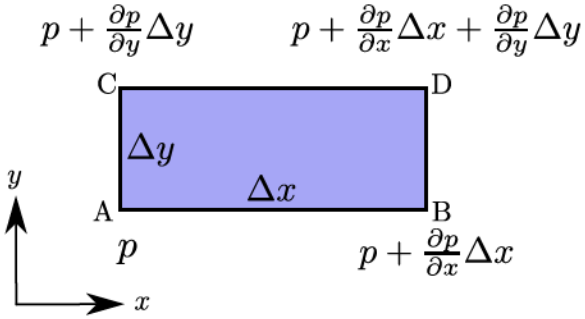


Figure 1.1 – Un rectangle de fluide

Euler suppose alors que la pression au point A est p (en Pa) et en déduit les valeurs de la pression pour chaque coin du rectangle figure 1.1. Une pression de un pascal correspond à une force de un newton exercée sur une surface de un m^2 : $1Pa = 1N \cdot m^{-2}$. La force de pression agissant sur chaque face est approximée en prenant la moyenne des pressions présentes au niveau de chaque sommet et en la multipliant par la surface. Ceci est illustré dans la figure 1.2. On peut donc en déduire les composantes x et y des forces.

$$F_x = F_{AC} - F_{BD} = -\frac{\partial p}{\partial x}\Delta y \Delta x \tag{1.12}$$

$$F_y = F_{AB} - F_{CD} = -\frac{\partial p}{\partial y}\Delta x \Delta y \tag{1.13}$$

Ensuite on applique la deuxième loi de Newton, c'est-à-dire la relation fondamentale de la dynamique :

$$\sum \vec{F} = M\frac{d\vec{v}}{dt}. \tag{1.14}$$

Pour chaque composante on se retrouve avec :

$$F_{g,x} + F_x = M_L\frac{dv_x}{dt}, \tag{1.15}$$

$$F_{g,y} + F_y = M_L\frac{dv_y}{dt}, \tag{1.16}$$

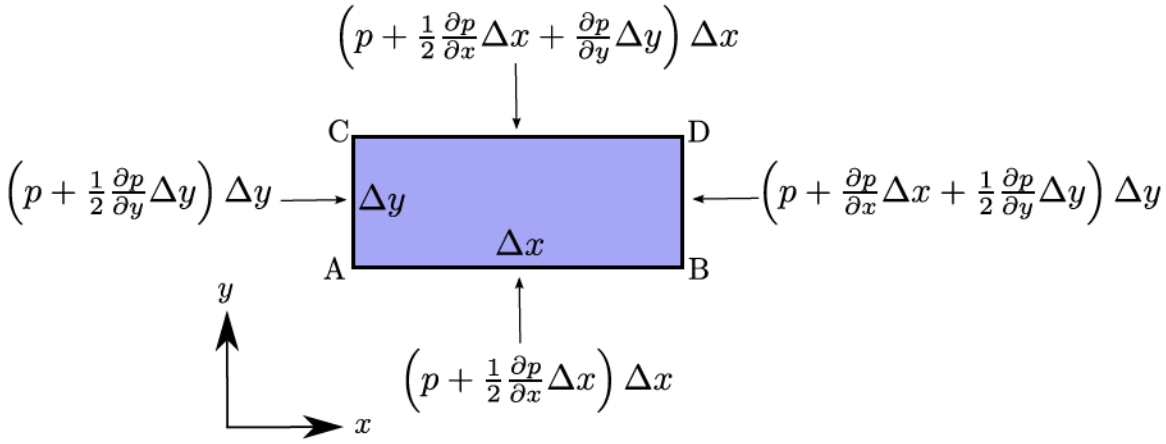


Figure 1.2 – Comportement de la force de pression sur le rectangle de fluide

$$-\frac{\partial p}{\partial x}\Delta x\Delta y = \rho\Delta x\Delta y\left(\frac{\partial v_x}{\partial x}v_x + \frac{\partial v_x}{\partial y}v_y + \frac{\partial v_x}{\partial t}\right), \tag{1.17}$$

$$-\rho g\Delta x\Delta y - \frac{\partial p}{\partial y}\Delta x\Delta y = \rho\Delta x\Delta y\left(\frac{\partial v_y}{\partial x}v_x + \frac{\partial v_y}{\partial y}v_y + \frac{\partial v_y}{\partial t}\right). \tag{1.18}$$

Ceci nous donne l'équation suivante de conservation du moment d'Euler pour un fluide incompressible (en deux ou trois dimensions) :

$$\frac{\partial \vec{v}}{\partial t} + \vec{v}\cdot\nabla\vec{v} = -\frac{1}{\rho}\nabla p + \vec{g}. \tag{1.19}$$

1.1.2 L'équation de conservation de la masse d'Euler

L'équation de conservation de la masse souvent appelée l'équation de continuité («continuity equation» en anglais) est l'expression d'un principe fondamental. Cette équation précise que la masse du fluide est conservée localement lorsque le fluide est incompressible : en lagrangien ou d'un point de vue particulaire, les particules de fluide qui entrent dans une région (avec une vitesse initiale non nulle) doivent en sortir sans perte ni création de particules. Pour obtenir cette équation, nous considérons un rectangle de fluide qui devient en trois dimensions un volume de contrôle, soit un cube de fluide figure 1.3. La conservation de la masse exige que la différence de flux net qui traverse le volume de contrôle soit nulle. En d'autres termes, il ne peut pas y avoir d'accumulation ou de disparition du fluide dans un volume de contrôle.

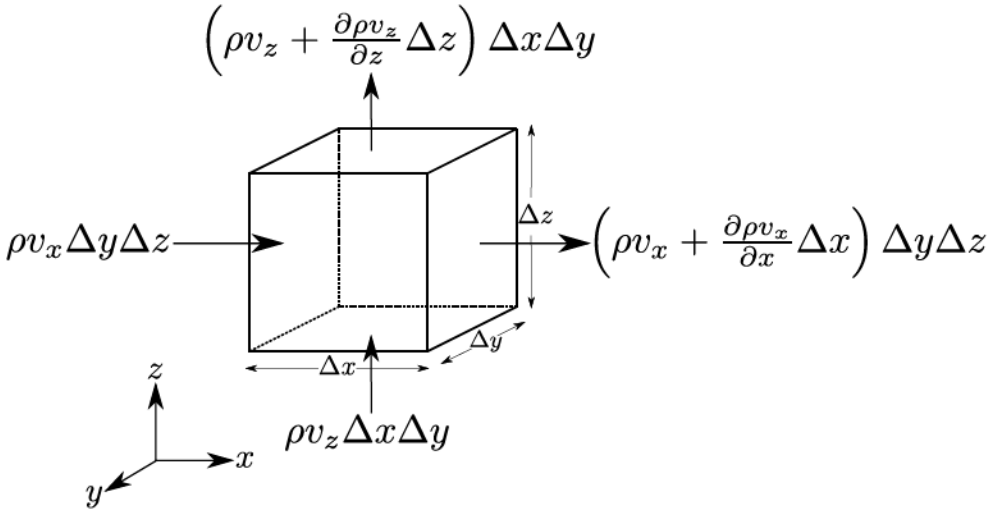


Figure 1.3 – Volume de contrôle

On considère une région arbitraire mais fixe du domaine Ω. La masse totale du fluide M est donc la suivante :

$$\begin{aligned} M &= \rho V, \\ M &= \iiint_{\Omega} \rho \, dV, \end{aligned} \tag{1.20}$$

ceci nous permet de simplifier pour le volume de contrôle figure 1.3, et d'en déduire la masse locale M_L :

$$M_L = \rho \Delta x \Delta y \Delta z. \tag{1.21}$$

L'évolution de la masse par rapport au temps, au sein du volume de contrôle est donnée par l'équation (1.22).

$$\frac{\partial \rho}{\partial t}\Delta x\Delta y\Delta z \tag{1.22}$$

À l'aide de figure 1.3, on peut calculer la quantité de fluide (ou flux) qui traverse le volume de contrôle. En particulier pour la face x, le flux est donné par :

$$\left(\rho v_x + \frac{\partial \rho v_x}{\partial x}\Delta x\right)\Delta y\Delta z - \rho v_x \Delta y\Delta z = \frac{\partial \rho v_x}{\partial x}\Delta x\Delta y\Delta z. \tag{1.23}$$

De la même façon, on a :

$$\frac{\partial \rho v_y}{\partial y}\Delta x\Delta y\Delta z \quad \text{pour la face } y, \tag{1.24}$$

$$\frac{\partial \rho v_z}{\partial z}\Delta x\Delta y\Delta z \quad \text{pour la face } z. \tag{1.25}$$

Par définition, et en combinant les équations (1.23)-(1.24)-(1.25), on obtient l'équation de la conservation de la masse qui nous permet de généraliser sur l'ensemble du domaine Ω en trois dimensions :

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_x}{\partial x} + \frac{\partial \rho v_y}{\partial y} + \frac{\partial \rho v_z}{\partial z} = 0,$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0. \tag{1.26}$$

Remarque

Les contraintes présentes dans un fluide apparaissent sous forme de deux équations de conservation : une pour la masse et l'autre pour la quantité de mouvement. Il est important de noter que pour un fluide incompressible, la dérivée lagrangienne (1.27) (ou dérivée particulaire) de la densité est nulle $\frac{D\rho}{Dt} = 0$ [BMF07] :

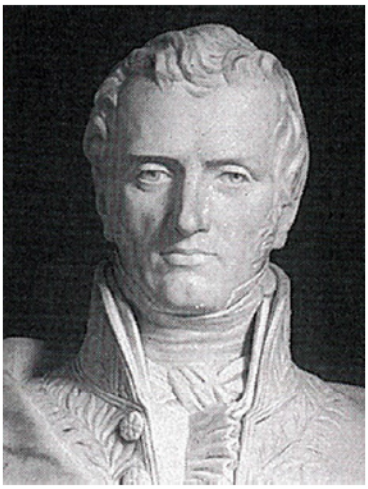
$$\frac{\partial \rho}{\partial t} + \vec{v} \cdot \nabla(\rho) = 0. \tag{1.27}$$

En soustrayant ceci à l'équation de conservation de la masse, on obtient $\rho \nabla \cdot \vec{v} = 0$ ce qui, bien souvent, nous conduit à une formulation plus simple :

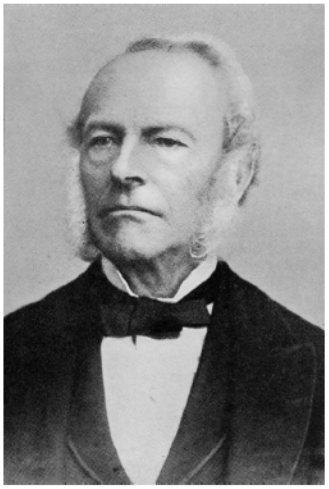
$$\nabla \cdot \vec{v} = 0. \tag{1.28}$$

Il est important de noter qu'un exposé sur la dérivée lagrangienne est fourni par la suite, au sein de la sous-section 1.4.2.

1.2 Les équations de Navier-Stokes



(a) Claude Louis Marie Henri Navier



(b) George Gabriel Stokes

Figure 1.4 – Portrait des physiciens Navier et Stokes

On doit à Henri Navier (mathématicien, ingénieur et économiste français ; figure 1.4a) l'idée, en 1822, d'introduire un terme supplémentaire à l'équation d'Euler, censé représenter une certaine perte d'énergie dans le fluide. En simplifiant son approche, on peut considérer qu'il a cherché à incorporer aux équations d'Euler, une équation dite de la chaleur. Cette dernière équation repose sur la loi de Fourier, établie mathématiquement par Jean-Baptiste Biot en 1804 puis expérimentalement par Fourier en 1822. La densité du flux de chaleur est proportionnelle au gradient de la température, si on note T la température d'un solide en kelvin (K), on a :

$$\vec{\phi} + \lambda \nabla T = 0, \tag{1.29}$$

où λ est un coefficient positif censé décrire le taux de dissipation de la chaleur et la densité de flux de chaleur $\vec{\phi}$ s'exprime en watt par mètre carré $(W \cdot m^{-2})$. Un bilan d'énergie et l'expression de la loi de Fourier conduisent à l'équation générale de conduction de la chaleur (1.30). En considérant nulle la production d'énergie au sein même du matériau, on obtient :

$$\rho \frac{\partial T}{\partial t} = \frac{\lambda}{c} \nabla^2 T, \tag{1.30}$$

où ρ est la densité du matériau et c est la chaleur spécifique massique du matériau en $J \cdot kg^{-1} \cdot K^{-1}$. En appliquant l'équation (1.30) pour le champ de vecteurs vitesse $\vec{v}$, on introduit au sein de l'équation de conservation du mouvement le terme de viscosité. Ainsi Navier, suivi par George Gabriel Stokes (mathématicien et physicien britannique ; figure 1.4b) en 1845, a proposé le modèle le plus utilisé en physique pour décrire l'évolution d'un fluide visqueux (ce terme rendant compte précisément de cette dissipation d'énergie sous forme de chaleur liée à la friction au sein du fluide). L'équation de conservation du mouvement et l'équation de conservation de la masse de Navier-Stokes s'écrivent de la manière suivante :

$$\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} = -\frac{1}{\rho} \nabla p + \vec{g} + \nu \nabla^2 \vec{v}, \tag{1.31}$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0, \tag{1.32}$$

où $\nu = \frac{\mu}{\rho}$ désigne la viscosité cinématique du fluide (unité SI : $m^2 \cdot s^{-1}$) et $\vec{v} \cdot \nabla \vec{v}$ est le terme d'advection (anglicisme) ou de convection.

1.3 Que signifie résoudre ces équations ?

La résolution des équations de Navier-Stokes fait partie de l'un des sept problèmes du millénaire proposés par l'institut de mathématiques Clay [Fef00].

Pour résoudre le problème de la fondation Clay, il convient de développer une théorie de résolution d'équations aux dérivées partielles. Pour développer une telle théorie, il faut tout d'abord définir ce que l'on entend par la résolution de ces équations. Ce problème [Fef00] bien posé, doit satisfaire plusieurs conditions dont trois d'entre elles sont les suivantes :

1. soit la condition d'existence : l'état du fluide étant supposé connu à un instant donné (initialisons ce temps à $t = 0$), il existe une solution à l'équation à des instants futurs $t + \Delta t$;
2. la condition d'unicité : il n'existe qu'une seule solution à l'équation coïncidant avec cet état initial ;

3. la condition de stabilité : cette solution est stable sous perturbations, ce qui signifie que si l'on modifie un tout petit peu l'état du fluide initial, les états ultérieurs ne seront que peu modifiés à leur tour, du moins pendant un certain temps.

Il y a deux manières de comprendre pourquoi l'équation de Navier-Stokes est compliquée : la vision mathématique et la vision physique. Pour le mathématicien, l'équation est compliquée parce que c'est une équation différentielle non-linéaire. Si vous la comparez à l'équation qui décrit le mouvement d'un ressort (ou à celle de la chaleur), la complication vient du terme $\vec{v} \cdot \nabla \vec{v}$ de convection qui est un terme non linéaire. Les turbulences observées auprès des réacteurs d'un avion sont l'illustration physique de la non-linéarité de l'équation de Navier-Stokes. Quand le problème a été posé par la fondation Clay, on savait déjà plusieurs choses au sujet de l'équation. D'une part, en deux dimensions, on sait démontrer qu'il existe toujours une solution. On sait également que si le champ de vecteurs vitesse initial est suffisamment petit, il existe toujours une solution régulière globalement définie. Physiquement, cela correspond aux régimes où on est sûr que l'écoulement sera laminaire (avec un faible nombre de Reynolds) et pas turbulent (nombre de Reynolds plus grand, supérieur à 2000), et donc on évite l'influence forte des non-linéarités. Grigori Perelman semble s'intéresser à la résolution de ces équations, après avoir démontré la conjecture Poincaré [CMM+10] en 2003 (également, un des sept problèmes du millénaire de la fondation Clay).

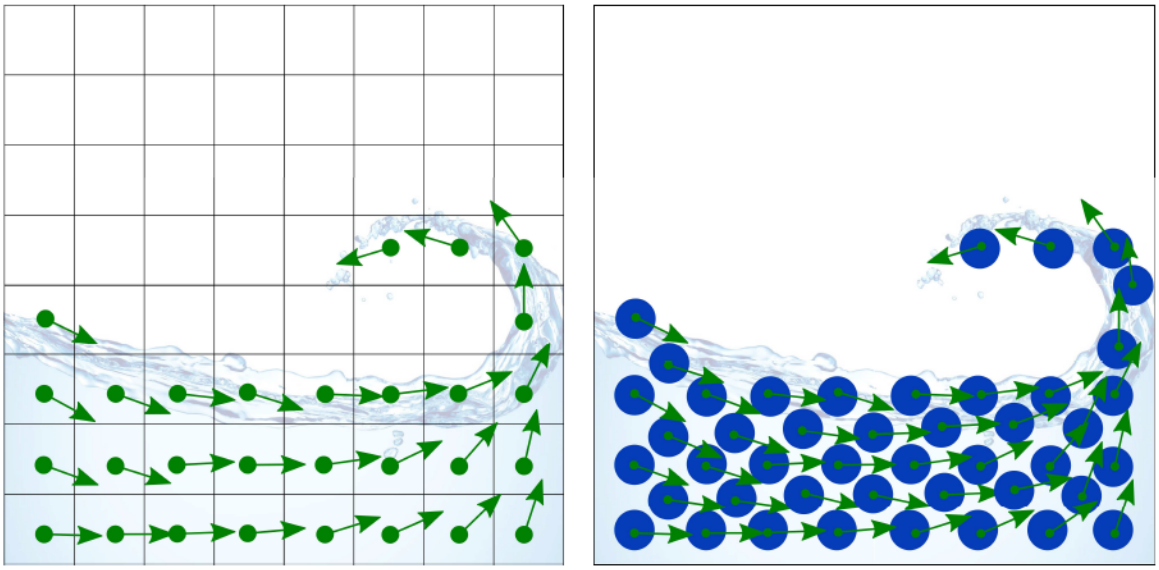
En informatique, en particulier pour la simulation de fluides, il est impossible d'implémenter l'équation exacte dans un ordinateur, c'est également le cas pour des équations aux dérivées ordinaires et partielles. On est obligé de remplacer ces dérivées par une approximation, il devient donc difficile d'obtenir une solution explicite des équations de Navier-Stokes. Pour cela, nous allons suivre deux grandes approches afin de décrire au mieux la dynamique des fluides.

1.4 Approches lagrangienne et eulérienne

Un fluide est un milieu matériel continu, déformable, qui peut s'écouler. En dynamique des fluides la description eulérienne et lagrangienne sont les deux techniques qui permettent de caractériser un écoulement. Nous allons décrire ces deux approches afin de mieux comprendre les enjeux de la résolution des équations de la dynamique des fluides (Euler/Navier-Stokes).

1.4.1 Comparaison des deux approches

Il existe deux grandes familles de discrétisation : l'approche eulérienne qui repose sur des grilles de calculs (figure 1.5a) et l'approche lagrangienne qui repose sur des particules (figure 1.5b). Les grilles discrétisent l'espace et les particules discrétisent la masse et également le domaine avec une approche différente. L'étape suivante consiste à obtenir des solutions approchées.



(a) Grille eulérienne (b) Particules lagrangiennes

Figure 1.5 – Illustrations des approches eulérienne et lagrangienne

Approche eulérienne

Elle décrit le champ de vitesses qui associe à chaque point un vecteur vitesse visible dans l'équation (1.33). La description eulérienne consiste donc à définir les grandeurs physiques en des points fixes du référentiel. Cette description est bien adaptée pour effectuer des analogies avec l'électromagnétisme (établissement d'équations locales). Dans cette description la vitesse du fluide est une fonction de deux variables indépendantes $\vec{r}$ et t :

$$\vec{v}(t) = \vec{v}(\vec{r}, t). \tag{1.33}$$

Plus généralement, il pourra être très utile de déterminer, en un point donné de l'espace, des caractéristiques du fluide telles que sa vitesse, sa pression, sa température. La description lagrangienne est peu adaptée à ce point de vue.

Approche lagrangienne

Contrairement à l'approche eulérienne, la photographie avec un temps de pose plus long permet de visualiser des trajectoires de la description lagrangienne (du nom du mathématicien Louis Lagrange). La description lagrangienne consiste donc à définir les grandeurs physiques en des points attachés à la matière : c'est la description utilisée en mécanique du point. Elle consiste donc à suivre dans le temps les particules du fluide le long de leurs trajectoires : c'est une description intuitive de leur mouvement. Pour une particule i, avec un vecteur position $\vec{r}_i$, on a pour un repère $(\vec{i}, \vec{j}, \vec{k})$:

$$\vec{v}_i(t) = \frac{d\vec{r}_i}{dt} = \frac{dx_i}{dt}\,\vec{i} + \frac{dy_i}{dt}\,\vec{j} + \frac{dz_i}{dt}\,\vec{k}. \tag{1.34}$$

Conclusion

La description eulérienne privilégie des points de l'espace auxquels on associe un champ de vitesses dépendant de l'espace et du temps : variables indépendantes. La description lagrangienne privilégie les particules que l'on suit dans leur déplacement et à qui on associe un ensemble de vitesses ne dépendant que du temps comme le montre la figure 1.5. À l'instant t, au point $\vec{r}$ en eulérien correspondant à la position $\vec{r}_i$ de notre particule lagrangienne on a :

$$\vec{v}_i(t)_{lagrange} = \vec{v}(\vec{r}, t)_{euler}. \tag{1.35}$$

1.4.2 Avantage : la dérivée lagrangienne

Soit $F(\vec{r},t)$ une grandeur attachée à une particule fluide (par exemple sa densité, sa vitesse...). $F(\vec{r},t)$ peut être une grandeur vectorielle ou scalaire. Entre t et $t+\Delta t$, la variation de F pour une même particule est :

$$DF = F(\vec{r}(t+\Delta t), t+\Delta t) - F(\vec{r},t) = \left(\frac{\partial F}{\partial t} + (\vec{v}\cdot\nabla)\,F\right)dt, \tag{1.36}$$

au premier ordre en t. On note $\frac{DF}{Dt}$ la dérivée particulaire ou dérivée en suivant le mouvement qui correspond à la dérivée de l'approche lagrangienne.

$$\frac{DF}{Dt} = \frac{dF}{dt} = \frac{\partial F}{\partial t} + \vec{v}\cdot\nabla F \tag{1.37}$$

Le terme $\frac{dF}{dt}$ désigne la dérivée totale ou ordinaire, $\frac{\partial F}{\partial t}$ représente la dérivée usuelle ou partielle par rapport au temps (pour une position fixe). On retrouve cette dérivée au sein des équations (1.7), (1.8). Dans le cas particulier ou F désigne la vitesse de notre particule de fluide, on peut montrer que :

$$\frac{D\vec{v}}{Dt} = \frac{\partial \vec{v}}{\partial t} + \vec{v}\cdot\nabla\vec{v}. \tag{1.38}$$

Il est donc très avantageux d'utiliser l'approche lagrangienne, plus précisément la dérivée lagrangienne, car le terme non-linéaire de l'équation de la dynamique des fluides est implicite. Afin de faire profiter les méthodes eulériennes de cet avantage, certaines techniques appelées méthodes hybrides sont nées. La méthode PIC/FLIP («Particle In Cell» et «Fluid-Implicit-Particle» en anglais) basée sur les écrits de Brackbill, [BKR88] et [BR86] est une des techniques les plus utilisées en simulation de fluides.

Tous les termes (sans l'advection), tels que l'accélération due à la gravité, la force de la pression, de viscosité et la résolution des conditions aux frontières sont intégrés sur la grille, tout comme dans une méthode eulérienne classique. Enfin, on interpole les valeurs calculées de vitesse et d'accélération de la grille sur les particules. On met ainsi à jour la position des particules présentes au sein du domaine, puis on ré-interpole les valeurs de vitesses des particules dans le champ vectoriel de vitesse

de la grille pour réaliser une certaine advection. Ces simulations offrent d'excellents résultats comme on peut le voir dans la figure 1.6.

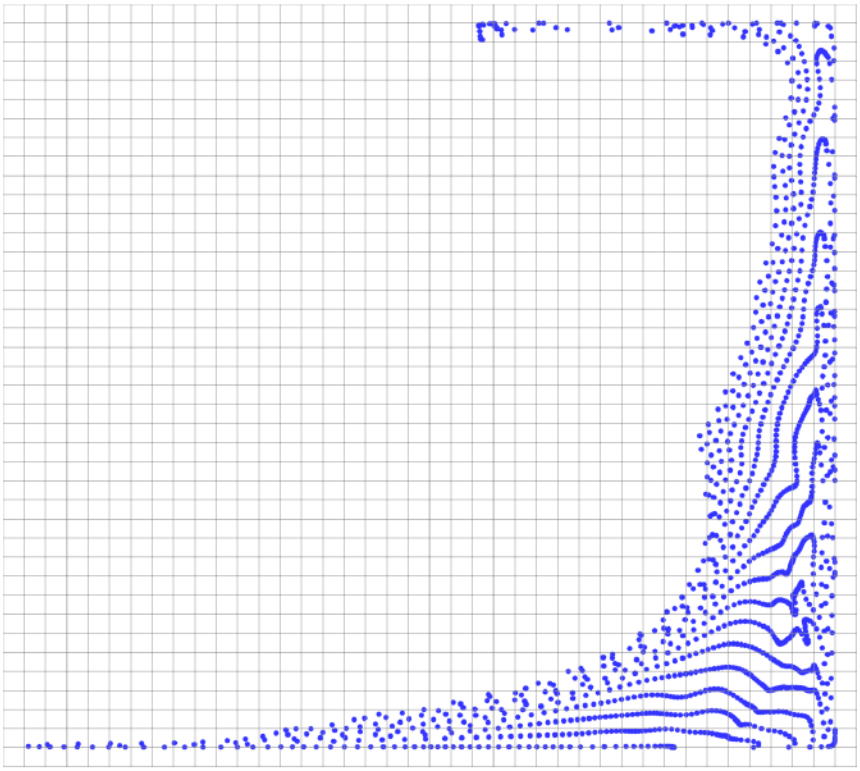


Figure 1.6 – Simulation d'un bris de barrage à l'aide de la méthode hybride PIC/FLIP

À cause de cela ce mémoire traitera essentiellement des méthodes dite purement lagrangiennes : les méthodes particulaires. Nous allons décrire, dans un premier temps les bases mathématiques de ces méthodes, puis nous allons les appliquer directement à la dynamique des fluides pour réaliser des simulations de qualité. Dans une optique de simulation temps réel, nous finirons sur le développement et l'application de ces algorithmes dans un contexte de programmation parallèle.

Chapitre 2

Particules hydrodynamiques lissées, ou Smoothed Particle Hydrodynamics (SPH)

La méthode SPH («Smoothed Particle Hydrodynamics» en anglais) est souvent présentée comme une méthode particulaire pure (lagrangienne). Publiée en 1977 par Lucy [Luc77], ainsi que Gingold et Monaghan [GM77], elle est d'abord appliquée à la modélisation de galaxies en astrophysique. L'approche particulaire se montre particulièrement adaptée à ce type de modélisation car une particule suffit à représenter un corps céleste. Cette méthode, alors très proche de la méthode de Monte-Carlo, consiste à estimer (représentation d'intégrale finie) des fonctions (scalaires ou vectorielles) ou leurs dérivées au moyen d'un ensemble fini de points de prélèvement (voisins). Très populaire, cette méthode fut adoptée en infographie [MCG03] afin d'étudier et d'observer le comportement des fluides à surface libre [Mon92, MK99]. Comparée à d'autres méthodes bien connues pour l'approximation numérique de dérivées, comme la méthode des différences finies, qui exige que les particules soient alignées sur une grille régulière, SPH peut estimer les dérivées continues à l'aide des différences de position d'un voisinage de particules disposé irrégulièrement.

L'ensemble des informations présentées dans cette section est extrait des publications suivantes [GM77, GM82, Mon92, Mon94, MK99, Mon05, Luc77, LL03, LL10]

dont les auteurs comptent parmi les plus reconnus pour la méthode SPH. Comme pour le chapitre 1, avant de lire ce qui suit, il est important de prendre connaissance du contenu de l'annexe A.

2.1 Principe fondamental des méthodes particulaires

On s'intéresse à une fonction f de la variable $\vec{r}$, qui est un vecteur qui représente une position (plus de détails dans la sous-section A.1.1), $f(\vec{r})$. Cette fonction peut être à valeurs complexes. L'idée principale est de remplacer cette fonction par une formulation intégrale équivalente en utilisant les propriétés du delta de Dirac (par le biais d'une convolution), (δ_{dirac}) obtenues par Paul Dirac en 1958 :

$$f(\vec{r}) = (f * \delta_{dirac})(\vec{r}) = \int_{\Omega} f(\vec{r}')\delta_{dirac}(\vec{r} - \vec{r}')d\vec{r}'. \tag{2.1}$$

Pour donner un sens à un objet δ_{dirac} vérifiant (2.1), il faut généraliser la notion de fonction ; on parle alors de fonctions généralisées ou de distributions. Pour plus de détails sur les distributions, se référer à [BCL99].

2.1.1 Principe en une dimension

f est définie en x et continue sur Ω . Ensuite, la fonction δ de Dirac ou encore la distribution de Dirac est remplacée par une fonction généralisée de lissage, appelée noyau, W avec un support fini $h > 0$.

$$\langle\ f(x)\ \rangle = \lim_{h\to 0} \int_{\Omega} f(x')W(x - x', h)dx'. \tag{2.2}$$

En statistique, cette formulation est appelée l'estimation par noyau (ou encore méthode de Parzen-Rosenblatt). Il s'agit d'une convolution entre la fonction f et la fonction noyau W (2.1). Toutefois, pour que cette égalité soit vraie il faut que la fonction noyau respecte certaines conditions :

– Son intégrale doit être normalisée,ce qui est appelée alors condition d'unicité :

$$\int_{\Omega} W(x,h)dx = 1. \tag{2.3}$$

– Elle doit converger (au sens des distributions) vers la distribution de Dirac :

$$\lim_{h\to 0} W(x,h) = \delta_{dirac}(x). \tag{2.4}$$

– Elle doit être symétrique ou paire :

$$W(x,h) = W(-x,h). \tag{2.5}$$

– Elle doit être à support compact ou fini :

$$W(x,h) = 0 \text{ si } \ |x| > h. \tag{2.6}$$

– Pour des raisons d'ordre physique, elle doit être non négative.

La plupart des fonctions noyaux utilisées en SPH sont des approximations d'une fonction noyau de type gaussienne. Une fonction noyau qui respecte l'équation (2.2) est aussi appelée, en analyse fonctionnelle, un noyau reproduisant. Une discussion sur les fonctions noyaux est présentée dans la section 2.3.

Enfin, pour obtenir une convergence d'ordre k, c'est à dire reproduire un polynôme de degré k ou inférieur, les fonctions noyaux doivent également satisfaire une condition supplémentaire :

$$\int_{\Omega} x^j W(x,h)dx = 0 \quad \text{avec} \quad 0 < j \leq k. \tag{2.7}$$

Ceci peut être vérifié en utilisant le développement en série de Taylor. Tout d'abord, on introduit la notion d'indices multiples :

$$|\alpha| = \alpha_1 + \alpha_2 + \cdots + \alpha_k, \quad \alpha! = \alpha_1!\alpha_2!\cdots\alpha_k!, \quad x^{\alpha} = x_1^{\alpha_1}x_2^{\alpha_2}\cdots x_k^{\alpha_k}. \tag{2.8}$$

pour $\alpha \in \mathbb{N}^k$ et $x \in \mathbb{R}^k$. On peut en déduire, si toutes les dérivées partielles de f sont continues jusqu'à l'ordre k, la notation d'indices multiples de l'opérateur de dérivée :

$$D^\alpha f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdots \partial x_k^{\alpha_k}}. \tag{2.9}$$

On écrit le développement en série de Taylor d'une fonction f de plusieurs variables dérivables $k+1$ fois et définie sur notre domaine Ω, centré en x' de la façon suivante en utilisant la notation d'indices multiples :

$$\begin{cases} f(x') = \sum\limits_{|\alpha| \leq k} \dfrac{D^\alpha f(x)}{\alpha!}(x'-x)^\alpha + \sum\limits_{|\alpha|=k+1} R_\alpha (x'-x)^\alpha, \\ \text{avec } |R_\alpha| \leq \max\limits_{y \in \Omega} |\dfrac{D^\alpha f(y)}{\alpha!}|. \end{cases} \tag{2.10}$$

On définit alors un reste R_α. Soit, pour plus de compréhension, le développement en série de Taylor en une dimension suivant de la fonction f au voisinage d'un point x' du polynôme de degré k :

$$f(x') = f(x) + \frac{df(x)}{dx}(x'-x) + \frac{1}{2}\frac{d^2 f(x)}{dx^2}(x'-x)^2 + \cdots + O(x'-x)^{k+1}, \tag{2.11}$$

où $O((x'-x)^{k+1})$ représente l'ordre de convergence du développement. En utilisant pour $k=1$, les équations (2.2) et (2.11) on obtient :

$$\begin{aligned} \langle\, f(x)\,\rangle =& f(x) \lim_{h \to 0} \int_\Omega W(x-x',h)dx' \\ &+ \frac{df(x)}{dx} \lim_{h \to 0} \int_\Omega (x'-x) W(x-x',h)dx' \\ &+ O((x'-x)^2) W(x-x',h). \end{aligned} \tag{2.12}$$

En respectant ainsi l'ensemble les conditions (2.3), (2.5) et (2.7), on obtient donc :

$$\langle\, f(x)\,\rangle = f(x) + O((x'-x)^2) W(x-x',h), \tag{2.13}$$

où $(x' - x)^2$ est l'erreur relative de l'approximation d'un polynôme de degré 1. Cette erreur relative est au maximum $O(h^2)$ car W dispose d'un support fini $h > 0$ (2.6). [Sam14], [LL03], [Mon05] et [CESM11] offrent plus de détails sur les conditions d'approximation.

2.1.2 Généralisation en deux et trois dimensions

À partir de l'estimation par noyau (2.2), on peut approximer à l'aide d'une somme finie utilisant la valeur de f définie sur des échantillons de points x_j ou encore des particules à une position $\vec{r}_j$ en trois dimensions.

$$\langle\ f(\vec{r})\ \rangle \approx \sum_j f(\vec{r}_j) W(\vec{r} - \vec{r}_j, h) V_j, \tag{2.14}$$

V_j est le volume occupé par la particule j et représente le pas de discrétisation. En SPH, son volume est donné par $V_j = \frac{m_j}{\rho_j}$ où m_j est la masse associée à la particule j et ρ_j est la densité de la distribution de particules au voisinage de celle-ci.

W dispose d'un support compact, ce qui veut dire qu'une partie de la sommation sur l'ensemble du domaine en (2.14) est égale à zéro. On se retrouve à réaliser une sommation sur un voisinage de n particules. En considérant h suffisamment petit, on peut écrire l'approximation suivante :

$$f(\vec{r}) = \sum_j^n f(\vec{r}_j) W(\vec{r} - \vec{r}_j, h) V_j. \tag{2.15}$$

De plus, des problèmes interviennent sous cette formulation, car elle ne respecte pas la propriété du delta de Kronecker $\sum V_j W(\vec{r} - \vec{r}_j, h) = \delta_{i,j}$. De nombreuses améliorations de la formulation SPH ont été proposées afin de corriger ce type d'erreur. Les problèmes et améliorations seront détaillés au sein des chapitre 3 et chapitre 4.

2.2 Principe des dérivées

Nous profitons de cette section pour faire référence à l'annexe A, dans laquelle de nombreux rappels mathématiques sont décrits (gradient, Laplacien ou encore diver-

gence en une, deux et trois dimensions).

2.2.1 L'opérateur gradient

Pour exprimer des équations différentielles à l'aide de la méthode SPH, il est nécessaire d'utiliser des approximations appropriées. Pour dériver une formulation SPH, il faut dériver la fonction $\vec{f}$ vectorielle dans l'équation (2.2).

$$\nabla\langle\ \vec{f}(\vec{r})\ \rangle = \lim_{h\to 0}\int_{\Omega}\nabla\vec{f}(\vec{r}')W(\vec{r}-\vec{r}',h)d\vec{r}', \tag{2.16}$$

et l'intégration par parties donne :

$$\nabla\langle\ \vec{f}(\vec{r})\ \rangle = \lim_{h\to 0}\left[\vec{f}(\vec{r}')W(\vec{r}-\vec{r}',h)\right]_{d\Omega} - \lim_{h\to 0}\int_{\Omega}\vec{f}(\vec{r}')(-1)\nabla W(\vec{r}-\vec{r}',h)d\vec{r}'. \tag{2.17}$$

Le théorème du flux-divergence, appelé aussi théorème de Green-Ostrogradski, affirme l'égalité entre l'intégrale de la divergence d'un champ vectoriel sur un volume Ω dans $\mathbb{R}^3$ et le flux de ce champ à travers la frontière $\mathcal{S}$ dans $\mathbb{R}^2$ du volume (qui est une intégrale de surface). Ce théorème est seulement applicable en trois dimensions. L'égalité est la suivante :

$$\iiint_{\Omega}\nabla\cdot\vec{F}d\Omega = \oiint_{\mathcal{S}}(\vec{F}\cdot\vec{n})dS. \tag{2.18}$$

L'usage du théorème du flux-divergence (2.18) est pour faciliter la démonstration et la compréhension, pour plus de détails sur l'obtention exacte de l'approximation de l'opérateur gradient à n-dimensions se référer à [Sam14]. L'équation (2.17), avec l'usage du théorème du flux-divergence, devient :

$$\nabla\langle\ \vec{f}(\vec{r})\ \rangle = \lim_{h\to 0}\left[\vec{f}(\vec{r}')W(\vec{r}-\vec{r}',h)\cdot\vec{n}\right]_{dS} + \lim_{h\to 0}\int_{\Omega}\vec{f}(\vec{r}')\nabla W(\vec{r}-\vec{r}',h)d\vec{r}'. \tag{2.19}$$

Le terme d'intégration de la surface s'annule (2.20), car W est à support compact donc $W(h,h)=0$:

$$\left[\vec{f}(\vec{r}')W(\vec{r}-\vec{r}',h)\cdot\vec{n}\right]_{dS} = 0. \tag{2.20}$$

Cela peut justement conduire à des erreurs, car dans le cas d'une simulation de fluide, la frontière du fluide ne satisfait pas toujours cette hypothèse. On peut le constater dans la figure 2.1. La figure 2.1a satisfait parfaitement l'égalité en (2.20) alors que cette égalité n'est pas vraie dans la figure 2.1b.

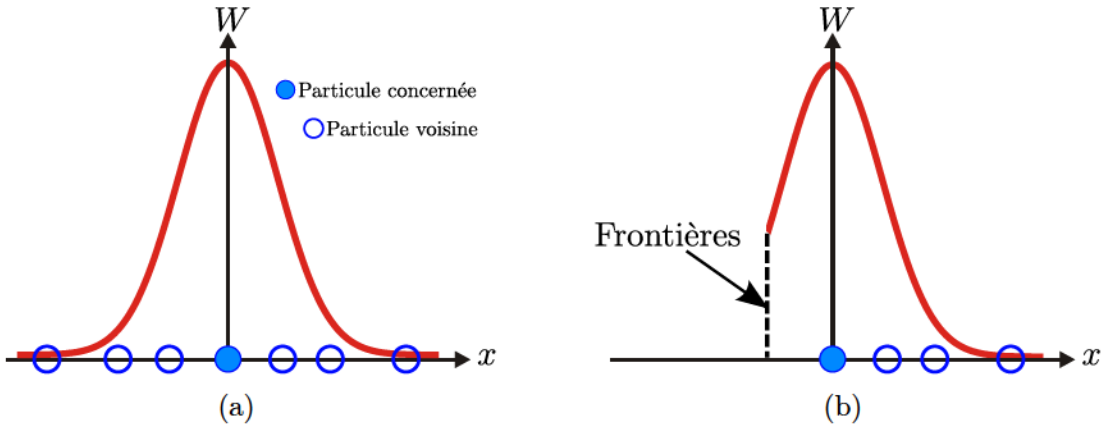


Figure 2.1 – Approximations SPH dans le cas unidimensionnel.
(a) Approximation pour une particule avec une distribution de particules voisines irrégulières. Cette illustration respecte (2.20).
(b) Approximation pour une particule dont le domaine est tronqué par la frontière. Cette illustration ne respecte pas (2.20).

L'équation (2.19) et l'approximation par somme donnent :

$$\nabla \vec{f}(\vec{r}) = \sum_{j}^{n} \vec{f}(\vec{r}_j) \nabla W(\vec{r} - \vec{r}_j, h) V_j. \tag{2.21}$$

Cette équation dispose toutefois d'un désavantage. On n'obtient pas nécessairement 0 dans le cas de l'approximation de la dérivée d'une fonction constante. Des conditions supplémentaires sont alors nécessaires dans le but d'effectuer une approximation correcte de la dérivée du premier ordre de f. On retrouve ces conditions dans les publications suivantes : [Sam14], [LL03] et [Mon05].

Remarque

En deux ou trois dimensions, on peut utiliser les coordonnées polaires ou les coordonnées sphériques ce qui permet réécrivant (2.19) d'obtenir certaines caractéristiques

de la fonction noyau W. En deux dimensions, on a $x = r\cos\theta$ et $y = r\sin\theta$:

$$W_{ij} = W(\vec{r}_i - \vec{r}_j, h) = W(\|\vec{r}_i - \vec{r}_j\|, h) = W(r_{ij}, h), \tag{2.22}$$

$$\nabla W_{ij} = \frac{\vec{r}_i - \vec{r}_j}{r_{ij}} \frac{\partial W(\vec{r}_i - \vec{r}_j, h)}{\partial r_{ij}} = \frac{\vec{r}_{ij}}{r_{ij}} \frac{\partial W_{ij}}{\partial r_{ij}}, \tag{2.23}$$

où r_{ij} est la distance Euclidienne séparant la particule concernée i et sa particule voisine j. Les équations (2.22), (2.23) se généralisent en trois dimensions.

2.2.2 Autres formulations des dérivées du premier ordre

∇f est souvent substitué par des formulations équivalentes que l'on peut retrouver au sein de la sous-section 3.1.3. Cependant, il existe d'autres usages de la dérivée du premier ordre ou encore la première dérivée, par exemple l'opérateur de divergence (on peut retrouver la description de cet opérateur dans la sous-section A.1.3) qui suit les mêmes règles que l'opérateur gradient c'est-à-dire :

$$\nabla \cdot \langle\, \vec{f}(\vec{r}) \,\rangle = \lim_{h\to 0} \int_\Omega \nabla \cdot \vec{f}(\vec{r}\,') W(\vec{r} - \vec{r}\,', h) d\vec{r}\,', \tag{2.24}$$

$$= \lim_{h\to 0} \int_\Omega \vec{f}(\vec{r}\,') \cdot \nabla W(\vec{r} - \vec{r}\,', h) d\vec{r}\,', \tag{2.25}$$

$$\text{d'où } \nabla \cdot \vec{f}(\vec{r}) = \sum_j^n \vec{f}(\vec{r}_j) \cdot \nabla W(\vec{r} - \vec{r}_j, h) V_j. \tag{2.26}$$

Il y a aussi le rotationnel (on peut retrouver la description de cet opérateur dans la sous-section A.1.4) qui suit également les mêmes règles que l'opérateur gradient et par conséquent l'opérateur de divergence :

$$\nabla \times \langle\, \vec{f}(\vec{r}) \,\rangle = \lim_{h\to 0} \int_\Omega \nabla \times \vec{f}(\vec{r}\,') W(\vec{r} - \vec{r}\,', h) d\vec{r}\,', \tag{2.27}$$

$$= \lim_{h\to 0} \int_\Omega \vec{f}(\vec{r}\,') \times \nabla W(\vec{r} - \vec{r}\,', h) d\vec{r}\,', \tag{2.28}$$

$$\text{d'où } \nabla \times \vec{f}(\vec{r}) = \sum_{j}^{n} \vec{f}(\vec{r}_j) \times \nabla W(\vec{r} - \vec{r}_j, h) V_j. \tag{2.29}$$

2.2.3 L'opérateur Laplacien

De la même manière que pour la première dérivée, on peut facilement obtenir la dérivée du second ordre, en l'occurrence le Laplacien. Il est cependant nécessaire d'obtenir des conditions d'approximation plus rigoureuses et strictes que pour la première dérivée. L'approximation de la deuxième dérivée est :

$$\nabla^2 f(\vec{r}) = \sum_{j}^{n} f(\vec{r}_j) \nabla^2 W(\vec{r} - \vec{r}_j, h) V_j. \tag{2.30}$$

La discrétisation de la dérivée du second ordre de l'équation (2.30) présente quelques problèmes selon Monaghan [Mon05]. Il existe des formulations différentes. La dérivée du second ordre est souvent utilisée pour calculer la viscosité ou encore la diffusion dans le fluide. Une alternative intéressante pour calculer la dérivée du second ordre est de combiner les différences finies et l'approximation SPH de la première dérivée. Tout d'abord, on se place en une dimension, ensuite on pose la formule suivante :

$$\int \frac{f(x') - f(x)}{x' - x} \frac{\partial W}{\partial x}(x - x', h) dx'. \tag{2.31}$$

Avec le développement en série de Taylor à une dimension de l'équation (2.11), on obtient en substituant $f(x')$ dans (2.31) :

$$\begin{aligned}
&\int \left(\frac{df}{dx}(x) + \frac{1}{2} \frac{d^2 f}{dx^2}(x)(x' - x) \right) \frac{\partial W}{\partial x}(x - x', h) dx' + o(h^3) \\
&\quad = \frac{df}{dx}(x) \int \frac{\partial W}{\partial x}(x - x', h) dx' + \frac{1}{2} \frac{d^2 f}{dx^2}(x) \int (x' - x) \frac{\partial W}{\partial x}(x - x', h) dx' + o(h^3), \\
&\quad = \frac{1}{2} \frac{d^2 f}{dx^2}(x) + o(h^3).
\end{aligned} \tag{2.32}$$

(2.32) est obtenue grâce à :

$$\int (x'-x)\frac{\partial W}{\partial x}(x'-x,h)dx' = [(x'-x)W(x-x',h)]_{-\infty}^{+\infty} - \int (-1)W(x-x',h)dx',$$
$$\int (x'-x)\frac{\partial W}{\partial x}(x-x',h)dx' = 1. \tag{2.33}$$

Donc on obtient en utilisant l'approximation SPH :

$$\frac{d^2 f}{dx^2}(x_i) = 2\sum_j^n \frac{f(x_i)-f(x_j)}{x_i-x_j}\frac{\partial W_{ij}}{\partial x} + o(h^3). \tag{2.34}$$

2.3 Choix du noyau d'interpolation employé

Il existe plusieurs formes de noyaux dans la littérature. Tous essayent de respecter l'ensemble des conditions décrites ci-dessus. Dans cette section, nous allons décrire les fonctions noyaux ou noyaux utilisés au sein de ce mémoire.

2.3.1 La gaussienne

Gingold et Monaghan [GM77] ont utilisé un noyau gaussien. Citons ici la première règle d'or de Joe Monaghan : *«If you want to find a physical interpretation then it is always best to assume the kernel is Gaussian»*. Ce noyau facile à utiliser est donc utilisé pour estimer la fonction f et ses deux premières dérivées en un point car il est indéfiniment dérivable.

$$W(\vec{r},h) = \beta_D \begin{cases} e^{-\frac{\|\vec{r}\|^2}{h^2}}, & \text{si } 0 \leq \|\vec{r}\| \leq h, \\ 0 & \text{sinon,} \end{cases} \tag{2.35}$$

où β_D représente la constante liée à la dimension D du problème (*cf.* tableau 2.1) afin de respecter les conditions ci-dessous, dans le cas d'une interpolation sur un champ de particules uniformément réparties. Il est à noter que cet ordre d'approximation n'est plus valable dès qu'elles ont abandonné cette répartition, d'où l'usage courant de techniques de (re)normalisation, présentées plus loin, afin de restaurer ces conditions (section 3.1.1).

β_{1D}	β_{2D}	β_{3D}
$\frac{1}{\sqrt{\pi}h}$	$\frac{1}{\pi h^2}$	$\frac{2h^3}{3\pi}$

Tableau 2.1 – Constantes du noyau gaussien en fonction de la dimension

2.3.2 Poly6

Müller [MCG03] introduit un noyau dit polynomial nommé Poly6. Ce noyau comporte la même forme qu'une gaussienne et répond aux mêmes conditions mathématiques : il est deux fois continûment différentiable et nous permet donc d'estimer la fonction f (à l'aide de W figure 2.2a) et ses deux premières dérivées (∇W figure 2.2b et $\nabla^2 W$ figure 2.2c). Ce noyau est fonction de r^2 et non de r, ce qui le rend pratique pour calculer la densité du fluide noté ρ_i liée à une particule i. Cette estimation sera décrite dans le chapitre 3.

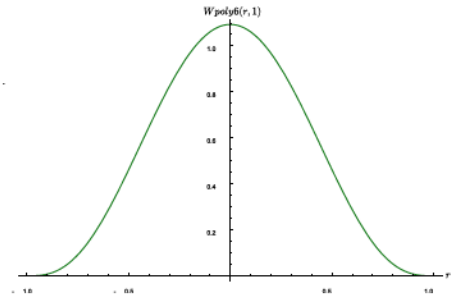
$$W_{poly6}(\vec{r}, h) = \beta_D \begin{cases} (h^2 - \|\vec{r}\|^2)^3 & \text{si } 0 \leq \|\vec{r}\| \leq h, \\ 0 & \text{sinon,} \end{cases} \tag{2.36}$$

où β_D représente la constante liée à la dimension D du problème *cf.* tableau 2.2.

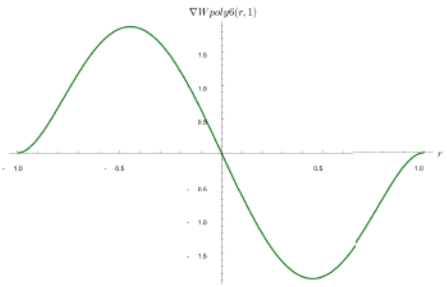
β_{1D}	β_{2D}	β_{3D}
$\frac{35}{32h^7}$	$\frac{4}{\pi h^8}$	$\frac{315}{64\pi h^9}$

Tableau 2.2 – Constantes du noyau Poly6 en fonction de la dimension

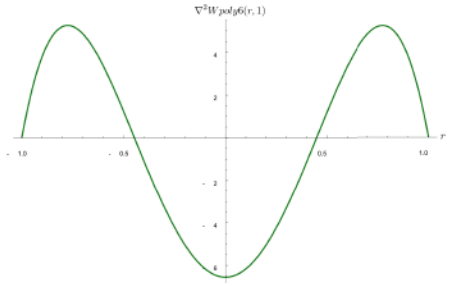
Ce type de noyau (polynomial) est très utilisé en simulation lagrangienne. Il en existe de nombreuses formes comme par exemple le noyau cubic spline (2.37), employé entre autre dans [MFZ97, Mon05, BT07, SP09, SB12] ce qui fait de lui le noyau le plus employé en simulation SPH. Pour l'obtention de β_D, il faut se référer à [LL10]. Ce noyau possède une distance de lissage de $2h$, il permet également d'approximer la



(a) poly6



(b) le gradient de poly6



(c) le Laplacien de poly6

Figure 2.2 – Graphe du noyau $W_{poly6}(r,h)$ pour $h=1$ et $\beta_{1D}=\frac{35}{32h^7}$

fonction f et ses deux premières dérivées.

$$W_{spline}(\vec{r},h)=\beta_D\begin{cases}1-\frac{3\|\vec{r}\|^2}{2h^2}+\frac{3\|\vec{r}\|^3}{4h^3} & \text{si } 0\leq\|\vec{r}\|\leq h,\\ \frac{1}{4}\left(2-\frac{\|\vec{r}\|}{h}\right) & \text{si } h\leq\|\vec{r}\|\leq 2h,\\ 0 & \text{sinon.}\end{cases} \tag{2.37}$$

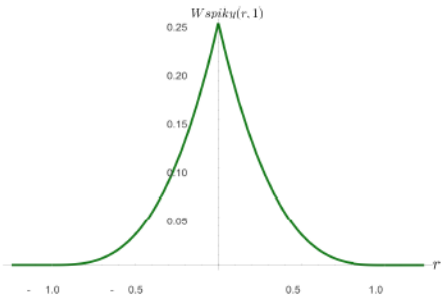
2.3.3 Spiky

Desbrun et Gascuel, [DG96], ont introduit un noyau un peu plus particulier :

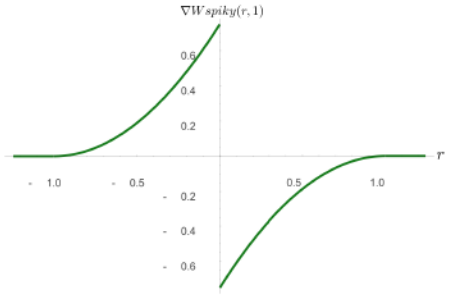
$$W_{spiky}(\vec{r},h)=\beta_D\begin{cases}(h-\|\vec{r}\|)^3 & \text{si } 0\leq\|\vec{r}\|\leq h,\\ 0 & \text{sinon.}\end{cases} \tag{2.38}$$

Il dispose d'une distance de lissage de h. Ce noyau est fonction de $\|\vec{r}\|$ et non de $\|\vec{r}\|^2$. Il est déconseillé de l'utiliser pour approximer la fonction f en l'occurrence la densité ρ_i à cause de la discontinuité en 0. L'emploi de ce noyau dans [MCG03, MM13] intervient seulement dans l'approximation de la première dérivée de la fonction f (à

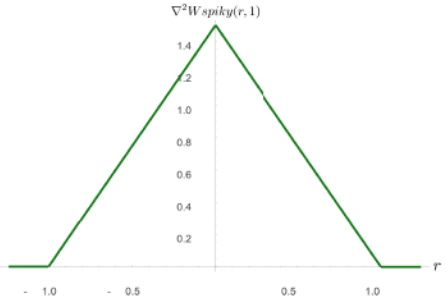
l'aide de ∇W figure 2.3b).



(a) spiky



(b) le gradient de spiky



(c) le Laplacien de spiky

Figure 2.3 – Graphe du noyau $W_{spiky}(r,h)$ pour $h=1$ et $\beta_{1D}=\frac{1}{4h^2}$

Le simple fait que le gradient du noyau n'est pas défini en 0 implique que l'usage de ce noyau pour estimer une deuxième dérivée est incorrect. Cette non-définition à l'origine permet justement d'obtenir une interprétation physique du gradient de la pression (défini dans la sous-section 3.1.3 du chapitre 3). Plus on se rapproche de l'origine, plus la valeur du gradient du noyau augmente. Ce contrairement au gradient d'une gaussienne où l'on observe une décroissance. Cette caractéristique du noyau permet d'éviter la formation de regroupement de voisins autour de la particule cible.

β_{1D}	β_{2D}	β_{3D}
$\frac{1}{4h^2}$	$\frac{10}{\pi h^5}$	$\frac{15}{\pi h^6}$

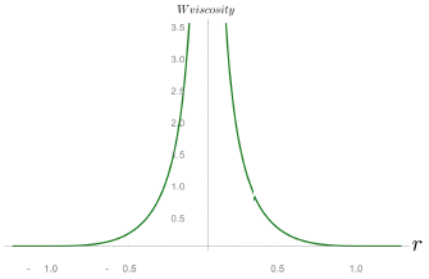
Tableau 2.3 – Constantes du noyau Spiky en fonction de la dimension

2.3.4 Viscosity

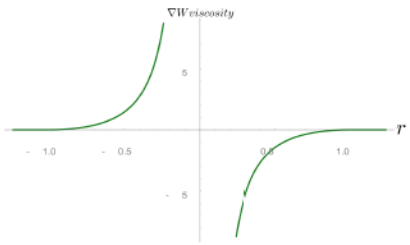
Müller, [MCG03], introduit un nouveau noyau nommé «Viscosity» (viscosité). Ce noyau, comme son nom l'indique, est utilisé afin de résoudre les problèmes où la viscosité intervient par le biais de dérivée seconde (Laplacien).

$$W_{viscosity}(\vec{r}, h) = \beta_D \begin{cases} -\frac{\|\vec{r}\|^3}{2h^3} + \frac{\|\vec{r}\|^2}{h^2} + \frac{h}{2\|\vec{r}\|} - 1 & \text{si } 0 \leq \|\vec{r}\| \leq h, \\ 0 & \text{sinon.} \end{cases} \tag{2.39}$$

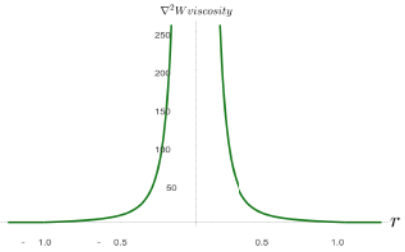
Comme on peut le remarquer dans le tableau 2.4 ce noyau n'est même pas utilisable



(a) viscosity



(b) le gradient de viscosity



(c) le Laplacien de viscosity

Figure 2.4 – Graphe du noyau $W_{viscosity}(\vec{r}, h)$ pour $h = 1$ et $\beta_{2D} = \frac{10}{3\pi h^2}$

β_{1D}	β_{2D}	β_{3D}
Indéfinie	$\frac{10}{3\pi h^2}$	$\frac{15}{2\pi h^3}$

Tableau 2.4 – Constantes du noyau Viscosity en fonction de la dimension

en une dimension (β_{1D}). Cependant, le choix de Müller de l'usage de ce noyau dans ces

simulations est dû à un aspect physique. Encore une fois ce noyau ne respecte que peu de conditions mathématiques pour l'application correcte d'un noyau dans le cas d'une interpolation SPH. Müller [MCG03] y voit une interprétation physique. Si l'on observe la forme du Laplacien figure 2.4c on constate que plus on approche de l'origine, plus la valeur du Laplacien du noyau est grande. Ceci confirme le comportement de viscosité : plus les particules voisines seront proches, plus elles vont avoir de l'importance dans le caractère diffus de la viscosité (visible dans la figure 3.4).

2.4 Conclusion

Au sein de ce chapitre, nous avons décrit les bases mathématiques de la méthode SPH, méthode purement lagrangienne. Elle nous permet d'estimer les dérivées continues d'une fonction à l'aide des différences de position d'un voisinage de points répartis irrégulièrement. Ainsi nous pouvons résoudre des équations différentielles comme celles de la dynamique des fluides. Le choix du noyau est important ; en effet comme décrit précédemment, certains noyaux se prêtent à l'estimation de certaines dérivées et d'autres non. Ceci est étudié dans le prochain chapitre.

Chapitre 3

Application des méthodes particulaires aux équations de la dynamique des fluides

À l'aide des outils mathématiques définis précédemment, de nombreux systèmes d'équations aux dérivées partielles ou ordinaires peuvent être résolus. Le but de ce mémoire est d'appliquer les méthodes particulaires à l'étude des simulations de fluide. Nous ne passerons pas en revue tous les modèles particulaires qui ont pu être développés dans les nombreux domaines de la physique. Nous allons seulement décrire quelques stratégies clefs de ces dernières années dans le domaine de l'infographie à l'aide de la méthode SPH. En approche lagrangienne, le fluide est traditionnellement considéré comme faiblement compressible. La raison est qu'il est nettement plus facile de calculer la pression à partir d'une équation d'état décrite dans la sous-section 3.1.2, plutôt que d'avoir à obtenir cette pression par le biais de la résolution d'une équation (à l'image des méthodes eulériennes qui utilisent une équation de Poisson). Ainsi au sein de ce chapitre, nous allons découvrir les méthodes et approches adoptées afin de rendre le fluide incompressible. Avant de lire ce qui suit, il est important de prendre connaissance du contenu de l'annexe B pour mieux appréhender les critères de stabilité ainsi que les schémas d'intégration.

3.1 Résolution des équations de la dynamique des fluides

Une vue conceptuelle de SPH est que l'écoulement du fluide est décomposé en portions de fluide. Chacune des portions i (particules) possède une masse associée (m_i), une densité (ρ_i), une vitesse ($\vec{v}_i$), une pression (p_i), *etc.* Ces valeurs sont constantes pour cette portion. La simplicité conceptuelle de ce modèle de Lagrange réside dans le fait que le fluide est décomposé en une série de particules, qui interagissent les unes avec les autres selon les lois de la dynamique des fluides décrites dans le chapitre 1. Pour les simulations décrites dans ce mémoire, il est préférable d'employer les équations d'Euler (section 1.1), qui disposent d'une formulation plus simple que les équations de Navier-Stokes (section 1.2) car elles ne font pas intervenir le terme de diffusion (viscosité) surligné dans l'équation (3.1). De plus quand le nombre de Reynolds (nombre sans dimension qui caractérise un écoulement) est beaucoup plus grand que l'unité, ce qui est presque toujours vrai dans les écoulements naturels, le terme proportionnel à la viscosité est négligeable, de sorte que les équations de la dynamique des fluides se réduisent aux équations d'Euler.

Rappel des équations de la dynamique des fluides

On note ρ_i, une valeur scalaire, la densité définie sur la particule i qui a pour position $\vec{r}_i$. L'influence d'une particule j sur une particule i dépend de sa distance et de son volume estimé. m_j, ρ_j et $\vec{r}_j$ sont la masse, la densité et la position de la particule j.

$$\rho\left(\frac{D\vec{v}}{Dt}\right) = -\nabla p + \mu\nabla^2\vec{v} + \rho\vec{g}. \tag{3.1}$$

Ceci permet de déduire directement l'accélération $\vec{a}_i$ et la position de la particule i concernée :

$$\vec{a}_i = \frac{D\vec{v}_i}{Dt} = \frac{d\vec{v}_i}{dt} = \frac{\sum \vec{F}_i}{\rho_i} = \sum \vec{f}_i, \tag{3.2}$$

$$\frac{D\vec{r}_i}{Dt} = \frac{d\vec{r}_i}{dt} = \vec{v}_i, \tag{3.3}$$

où $\vec{f}_i$ représente la force par unité de masse (accélération en $m \cdot s^{-2}$).

3.1.1 L'estimation de la densité

La masse de la particule (m_i) est définie par l'utilisateur. La densité, elle, représente un champ scalaire défini au sein du domaine, représenté par le fluide. À l'aide de l'équation (2.15), l'estimation de la densité peut être formulée de la façon suivante :

$$\begin{aligned} \{\rho\}_i &= \sum_{j=1}^{n} \frac{m_j}{\rho_j} \rho_j W(\vec{r}_i - \vec{r}_j, h), \\ \rho_i &= \sum_{j=1}^{n} m_j W_{ij}. \end{aligned} \tag{3.4}$$

Dans de nombreuses publications [LL03, Mon05, SP08, LL10], cette estimation (3.4) est considérée comme conservative c'est-à-dire qu'elle conserve la masse globalement. Considérant h constant, on peut intégrer la densité estimée de la façon suivante :

$$\int_{\Omega} \rho(\vec{r}) d\vec{r}' \approx \sum_{j} m_j = M. \tag{3.5}$$

La masse du fluide est contenue au sein de l'ensemble des particules définies, on obtient donc, à l'aide de (3.4), une estimation de la densité qui conserve la masse (3.5).

Une autre approche de l'estimation de la densité (3.9) au sein du fluide peut être obtenue à l'aide de l'équation de la conservation de la masse (décrite dans la sous-section 1.1.2). Cette équation ci-dessous peut également être obtenue en utilisant la dérivée totale de l'estimation de la densité définie en (3.4) :

$$\left\{ \frac{d\rho}{dt} \right\}_i = \sum_{j=1}^{n} m_j \left[\frac{\partial W_{ij}}{\partial \vec{r}_i} \cdot \frac{d\vec{r}_i}{dt} + \frac{\partial W_{ij}}{\partial \vec{r}_j} \cdot \frac{d\vec{r}_j}{dt} + \frac{\partial W_{ij}}{\partial h} \frac{dh}{dt} \right]. \tag{3.6}$$

Or on sait que :

$$\frac{d\vec{r}_i}{dt} = \vec{v}_i \ , \ \frac{d\vec{r}_j}{dt} = \vec{v}_j \ , \ \frac{dh}{dt} = 0. \tag{3.7}$$

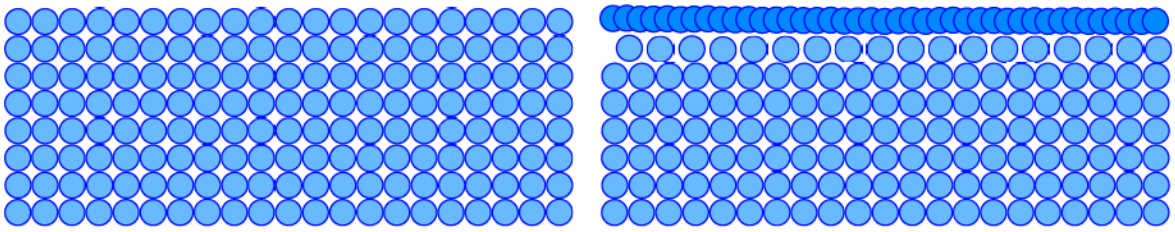
La valeur de h est constante car elle représente le support du noyau. La distance de lissage h en SPH est généralement fixée au début de la simulation. (3.7) nous permet d'écrire :

$$\left\{\frac{d\rho}{dt}\right\}_i = \sum_{j=1}^{n} m_j \left(\vec{v}_i \cdot \nabla_i W_{ij} + \vec{v}_j \cdot \nabla_j W_{ij}\right). \tag{3.8}$$

De plus, $\nabla_i W_{ij} = -\nabla_j W_{ij}$, grâce à la règle de dérivation en chaîne, ce qui nous donne :

$$\frac{d\rho_i}{dt} = \sum_{j=1}^{n} m_j \left(\vec{v}_i - \vec{v}_j\right) \cdot \nabla W(\vec{r}_i - \vec{r}_j, h). \tag{3.9}$$

Cette équation est généralement préférée à l'estimation de la densité à l'aide de l'équation (3.4), car elle permet d'éliminer les lacunes des particules situées sur la surface, visibles dans la figure 3.1. Elle permet également de plus facilement initialiser le fluide, étant donné que toutes les particules sont créées et initialisées avec leur densité égale à la densité de repos du fluide ρ_0 (utilisée dans la sous-section 3.1.2).



(a) (b)

Figure 3.1 – Distribution des particules proches de la surface du fluide.
(a) L'utilisation de l'équation (3.9) permet une meilleure distribution.
(b) L'utilisation de l'équation (3.4) impose aux particules situées à la surface de se rapprocher afin d'atteindre une densité constante.

Cependant, l'usage de l'équation (3.9) pour estimer la densité est nettement plus sujet aux problèmes de stabilité liés aux conditions CFL (Courant–Friedrichs–Lewy). Ces conditions sont décrites dans la section B.1. Cette estimation dépend fortement de la divergence des vélocités (vitesses) contrairement à l'estimation de l'équation (3.4).

Correction du noyau pour estimation de la densité

Dans la littérature, il existe une certaine correction utilisée sur l'équation (3.4) afin de palier les artefacts visibles dans la figure 3.1b. Cette correction appelée le filtre de Shepard, [She68], est une correction du noyau de l'ordre 0 :

$$\rho_i = \sum_{j=1}^{n} m_j W_{ij}^*, \tag{3.10}$$

$$\text{où} \qquad W_{ij}^* = \frac{W_{ij}}{\sum_{j=1}^{n} \frac{m_j}{\rho_j} W_{ij}}. \tag{3.11}$$

Cette correction appliquée périodiquement (appliquée tous les 10 ou 20 pas de temps) permet de normaliser le noyau. Et ainsi de fournir aux particules situées aux frontières une estimation de la densité plus proche de la densité cible ρ_0.

3.1.2 L'équation d'état

Une fois l'estimation de la densité obtenue, la pression correspondante peut être calculée. Pour ce faire, beaucoup d'auteurs utilisent les équations d'état («state equation» en anglais). Ces équations sont utilisées pour lier ensemble les propriétés du fluide, en particulier, pour déduire la pression liée à la densité. Comme l'équation d'état est arbitraire, différents choix peuvent être faits en fonction des besoins des différentes simulations.

La première équation utilisée est celle des gaz parfaits, dans cette équation on considère le fluide compressible et à température constante :

$$p_i = k\rho_i, \tag{3.12}$$

où k représente une constante définie par l'utilisateur. Une modification de cette équation a été proposée par [DG96] dans le but que chaque particule i atteigne une densité constante ρ_0 (souvent appelée densité au repos du fluide) :

$$p_i = k\left(\rho_i - \rho_0\right). \tag{3.13}$$

Cette formulation (3.13) est utilisée au sein de la simulation SPH de Müller [MCG03].

La seconde équation utilisée est obtenue par [Mon94] suite aux écrits de Batchelor, [Bat00], initialement en 1973. Cette deuxième équation (3.14) est appelée l'équation de Tait («Tait's equation» en anglais). L'équation a été publiée à l'origine par Peter Guthrie Tait en 1888 :

$$p_i = B\left(\left(\frac{\rho_i}{\rho_0}\right)^{\gamma} - 1\right), \tag{3.14}$$

où $B = \frac{\rho_0 k}{\gamma}$ est une constante, ρ_0 est la densité cible et γ est une constante, habituellement comprise entre 1 et 7. Morris et Fox [MFZ97] suggère d'utiliser un $\gamma = 1$ qui facilite l'obtention d'un faible nombre de Reynolds. Dans le but d'éviter les transitions qui font apparaître des instabilités dues à l'amplification des perturbations. Le choix de $\gamma = 1$ offre une version compressible du fluide, similaire à celle décrite dans l'équation (3.13), alors que le choix de fixer $\gamma = 7$ offre un modèle faiblement compressible (utilisé justement pour «Weakly compressible SPH for free surface flows» [BT07]). Cette équation de Tait est devenue une référence depuis 2003 car elle est utilisée pour presque toutes les simulations modernes SPH.

La valeur de la densité au repos ou cible est bien souvent choisie pour correspondre à la physique, c'est-à-dire $\rho_0 = 1000\ kg \cdot m^{-3}$ (la densité de l'eau au repos à 4 °C). Cependant, le choix de cette constante ρ_0, nous fournit des pressions non physiques, comme on peut le constater dans l'équation (3.13), car on obtient des valeurs de pressions négatives ou positives. On parlera donc, de pression relative. Le but de l'obtention de valeurs positives et négatives pour la pression relative sera décrit dans la prochaine section (sous-section 3.1.3).

L'avantage d'utiliser une équation d'état est qu'il n'est pas nécessaire de résoudre une loi de Poisson pour obtenir la pression (généralement utilisée dans les approches eulériennes et très longue à résoudre).

3.1.3 L'estimation du gradient de la pression

Munis de la pression, nous pouvons donc obtenir la force de pression au sein du fluide. L'estimation du gradient de la pression en notation SPH est la suivante :

$$\begin{aligned}\left\{-\frac{1}{\rho}\nabla p\right\}_i &= -\frac{1}{\rho_i}\nabla p_i, \\ &= -\frac{1}{\rho_i}\sum_{j=1}^{n}\frac{m_j}{\rho_j}p_j\nabla W(\vec{r}_i-\vec{r}_j,h).\end{aligned} \tag{3.15}$$

Cependant, il ne faut pas négliger la deuxième règle d'or de Joe Monaghan : *«Rewrite formulas with density inside operators»*. Pour cela on définit différentes formulations du gradient avec la densité. On retrouve une formulation différentielle du gradient décrite par Colin, Egli et Lin [CEL06] (pour plus de détails sur l'obtention de ces formulations *cf.* sous-section A.1.6) :

$$\begin{aligned}\left\{-\frac{1}{\rho}\nabla p\right\}_i &= \left\{-\frac{1}{\rho^2}\left(\nabla\left(\rho p\right)-p\nabla\rho\right)\right\}_i, \\ &= -\frac{1}{\rho_i^2}\sum_{j=1}^{n}m_j\left(p_j-p_i\right)\nabla W(\vec{r}_i-\vec{r}_j,h).\end{aligned} \tag{3.16}$$

Müller [MCG03] décrit une version similaire de l'équation précédente :

$$\left\{-\frac{1}{2\rho}\nabla\left(1\,p\right)\right\}_i = -\frac{1}{\rho_i}\sum_{j=1}^{n}m_j\frac{p_j+p_i}{2\rho_j}\nabla W(\vec{r}_i-\vec{r}_j,h), \tag{3.17}$$

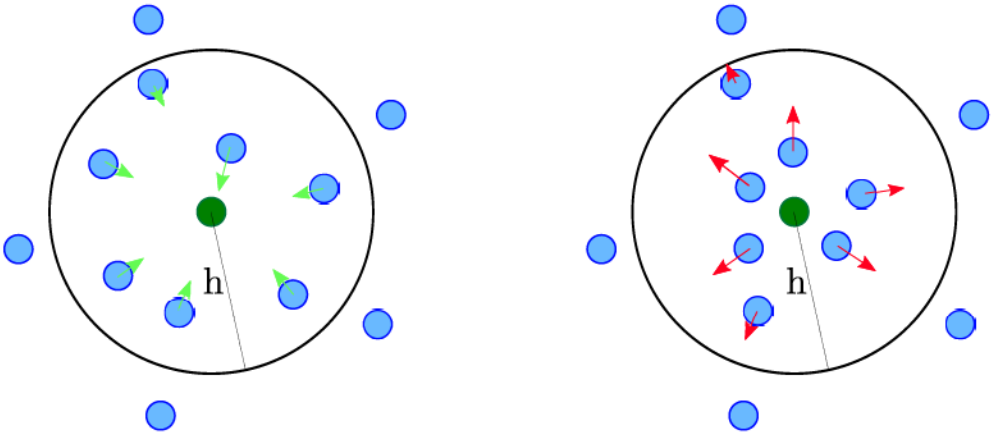
$$\left\{-\frac{1}{\rho}\left(\frac{1\,\nabla p+p\,\nabla 1}{2}\right)\right\}_i \neq \left\{-\frac{1}{\rho}\nabla p\right\}_i. \tag{3.18}$$

On constate que Müller ne réalise pas exactement la bonne estimation dans l'équation (3.18). Il tente plutôt de réaliser une sorte de moyenne arithmetique de la pression décrite dans l'équation (3.17). Malheureusement, ces formulations ne permettent pas de valider la troisième loi de Newton. Comme les pressions obtenues ne sont pas les mêmes d'une particule à l'autre, les forces de pression à obtenir doivent être symétriques afin de respecter la loi d'action-réaction de Newton.

Pour cela est apparue une formulation dite symétrique du gradient (3.19). Cette formulation décrite dans [Mon92, Mon05, LL03, CEL06] est devenue la référence pour estimer le gradient de la pression [BT07, SP09, MM13] :

$$\begin{aligned}\left\{-\frac{1}{\rho}\nabla p\right\}_i &= \left\{-\nabla\left(\frac{p}{\rho}\right) - \frac{p}{\rho^2}\nabla\rho\right\}_i, \\ &= -\sum_{j=1}^{n} m_j \left(\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}\right)\nabla W(\vec{r}_i - \vec{r}_j, h).\end{aligned} \tag{3.19}$$

À partir de cette estimation du gradient de la pression, on obtient une force de pression par unité de volume. Cette force dépend directement de la valeur de la pression obtenue à l'aide de l'équation d'état décrite dans la sous-section 3.1.2. Cette force illustrée au sein de la figure 3.2 varie en fonction de la valeur non physique de la pression. Une pression relative négative (une valeur de densité inférieure à la densité au repos de notre fluide) nous fournit une force d'attraction (figure 3.2a). Alors qu'une valeur de pression relative positive nous indique une densité supérieure à la densité cible de repos de notre fluide (figure 3.2b) et donc une force répulsive.



(a) Pression négative (densité insuffisante) (b) Pression positive (densité trop élevée)

Figure 3.2 – Comportement de la force de pression

3.1.4 L'estimation de la viscosité

C'est la viscosité du fluide qui assure, par l'intermédiaire de la force de friction qui s'exerce entre les différentes couches du fluide, le transport diffusif de la quantité de mouvement. La viscosité peut s'interpréter comme une diffusion de quantité de mouvement comme on peut le constater avec la présence du Laplacien dans les équations de Navier-Stokes. Le coefficient caractérisant cet aspect diffusif est le coefficient de viscosité dynamique μ. Il caractérise l'aptitude du fluide à s'écouler de façon visqueuse et s'exprime en pascal·seconde ($Pa \cdot s$).

On peut décrire la viscosité en suivant l'équation de Navier-Stokes [MCG03] ou encore utiliser un terme de dissipation artificiel appelé viscosité artificielle, comme proposé par [Mon92, Mon94] et appliqué au sein des simulations [BT07, SP09].

L'approximation de la viscosité selon Matthias Müller [MCG03]

En partant de l'approximation du Laplacien obtenue par Colin, Egli et Lin [CEL06] à l'aide de l'usage double de la formulation différentielle du gradient (*cf.* sous-section A.1.6 pour plus de détails), on a :

$$\nabla^2 f_i = \frac{1}{\rho_i} \sum_{j=1}^{n} m_j \left(f_j - f_i\right) \left(\nabla^2 W(\vec{r}_i - \vec{r}_j, h) - \frac{2}{\rho_i} \nabla W(\vec{r}_i - \vec{r}_j, h) \cdot \nabla \rho_i\right). \tag{3.20}$$

De plus, ρ_i est censé être égal à ρ_0 en tout temps (si on considère le fluide incompressible). ρ_i est idéalement une fonction constante et donc $\nabla \rho_i = 0$. Ceci nous permet d'écrire l'équation (3.20) de la façon suivante :

$$\left\{\frac{\mu}{\rho} \nabla^2 \vec{v}\right\}_i = \frac{\mu}{\rho_i^2} \sum_{j=1}^{n} m_j \left(\vec{v}_j - \vec{v}_i\right) \nabla^2 W(\vec{r}_i - \vec{r}_j, h). \tag{3.21}$$

Mathias Müller décrit donc la viscosité à partir de l'équation (3.21) :

$$\left\{\frac{\mu}{\rho} \nabla^2 \vec{v}\right\}_i = \frac{\mu}{\rho_i} \sum_{j=1}^{n} \frac{m_j}{\rho_j} \left(\vec{v}_j - \vec{v}_i\right) \nabla^2 W(\vec{r}_i - \vec{r}_j, h), \tag{3.22}$$

où μ est une constante appliquée à chaque particule, constante définie par l'utilisateur au sein de la simulation afin de fournir un effet visuel plus ou moins visqueux.

Remarque

La formulation et l'usage du Laplacien du noyau dans le but d'estimer la viscosité peut conduire à des erreurs. La forme du Laplacien est très (trop) sensible à l'organisation et au positionnement des particules. Pour plus d'informations sur les erreurs d'interpolation du second ordre au sein de la méthode SPH, se référer aux écrits de Monaghan [Mon05]. Pour pallier cette lacune, Müller utilise un noyau qui ne satisfait pas certaines conditions (mathématiques) essentielles (noyau Viscosity sous-section 2.3.4), mais qui respecte l'interprétation physique de la viscosité. De nombreux autres auteurs proposent et utilisent une estimation de la viscosité à l'aide d'une viscosité artificielle qui permet de conserver et d'utiliser des noyaux respectant les conditions d'approximations.

Viscosité artificielle

Comme dans de nombreuses méthodes numériques, une viscosité artificielle peut être ajoutée au modèle non-visqueux de l'équation d'Euler. Cette viscosité artificielle, désignée par le terme «Artificial Viscosity» (AV) en anglais et notée Π s'inscrit directement dans l'équation de conservation du mouvement d'Euler (et non Navier-Stokes). C'est un terme de stabilisation des calculs, utilisé au sein de méthodes eulériennes et introduit au sein des méthodes SPH par Monaghan ([MG83] et [Mon85]) pour éviter l'interpénétration des particules (visible sur la figure 3.3) lors du calcul de phénomènes de chocs.

$$\frac{D\vec{v}_i}{Dt} = \vec{g} - \frac{1}{\rho_i}\nabla p_i + \begin{cases} \sum_{j=1}^{n} m_j \Pi_{ij} \nabla W(\vec{r}_{ij}, h) & \text{si } \vec{v}_{ij} \cdot \vec{r}_{ij} < 0, \\ 0 & \text{si } \vec{v}_{ij} \cdot \vec{r}_{ij} \geqslant 0. \end{cases} \tag{3.23}$$

Ainsi Monaghan propose de prendre une forme symétrique, inspirée des différences finies. On peut démontrer [Mon05] que $\vec{v}_{ij} \cdot \vec{r}_{ij} > 0$ est équivalent à $\nabla \cdot \vec{v} > 0$. En utilisant l'équation de l'estimation de la divergence SPH (2.26) et la définition du gradient du noyau (2.23), le produit scalaire $\vec{v}_{ij} \cdot \vec{r}_{ij}$ reproduit le comportement de la divergence dans le cas des méthodes particulaires (avec $\vec{v}_{ij} = \vec{v}_i - \vec{v}_j$). Sur la figure 3.3, on constate comment l'équation (3.23) est appliquée.

$\vec{v}_{ij} \cdot \vec{r}_{ij} > 0$ (a) Divergence positive

$\vec{v}_{ij} \cdot \vec{r}_{ij} < 0$ (b) Divergence négative

Figure 3.3 – Comportement de la divergence à l'aide de particules

Π_{ij} est défini de la manière suivante :

$$\Pi_{ij} = \nu \left(\frac{\vec{v}_{ij} \cdot \vec{r}_{ij}}{\|\vec{r}_{ij}\|^2 + \epsilon h^2} \right), \tag{3.24}$$

où ϵh^2, avec $\epsilon = 0.01$, placé au dénominateur a pour but d'éviter que Π_{ij} soit indéfini, quand $\vec{r}_{ij} = 0$. Ceci arrive quand deux particules sont trop proches l'une de l'autre. La viscosité cinématique ($\nu = \frac{\mu}{\rho}$) s'écrit de la façon suivante :

$$\nu = \frac{2\alpha_{AV} h \|\vec{v}_{ij}\|}{\rho_i + \rho_j}, \tag{3.25}$$

α_{AV} définit l'influence de cette viscosité artificielle. Pour ne pas introduire trop de dissipation, sa valeur est généralement choisie : $0.02 \leq \alpha_{AV} \leq 0.5$. Selon Monaghan cette approximation de la viscosité conserve l'aspect linéaire et angulaire de la conservation du mouvement.

Viscosité artificielle réaliste

En s'inspirant d'une expression SPH modélisant la conduction thermique, Morris dans son article [MFZ97], applique le principe de viscosité artificielle de Monaghan, avec une viscosité dynamique (μ) propre à chaque particule :

$$\left\{ \left(\frac{1}{\rho} \nabla \cdot \mu \nabla \right) \vec{v} \right\}_i = \sum_{j=1}^{n} \frac{m_j \left(\mu_i + \mu_j \right) \vec{r}_{ij} \cdot \nabla W_{ij}}{\rho_i \rho_j \left(\vec{r}_{ij}^2 + \epsilon h^2 \right)} \vec{v}_{ij}. \tag{3.26}$$

Cette nouvelle formulation de la viscosité artificielle conserve parfaitement l'aspect linéaire et angulaire de la conservation du mouvement. Plus réaliste, cette nouvelle viscosité est mieux adaptée aux écoulements de fluide à faible nombre Reynolds dit laminaire (très peu turbulents, nombre Reynolds inférieur à 2000) comme l'a remarqué Morris. L'apport de cette forme par rapport à celle de Monaghan est de projeter les composantes scalaires de $\vec{v}$ sur le noyau, et non le vecteur. Le gain en précision sur la disposition initiale est donc sensible. Par abus de langage cette nouvelle viscosité est appelée viscosité laminaire («laminar viscosity» en anglais) pour différencier la viscosité artificielle de Monaghan, de celle de Morris, car les deux reposent sur le même principe.

Remarque

L'usage d'une viscosité artificielle ou laminaire permet donc d'obtenir, en une seule étape de calcul, la valeur du gradient de la pression et un terme de dissipation (viscosité) car on utilise le même gradient du noyau pour estimer les deux. Ceci permet d'accroître les performances de calcul.

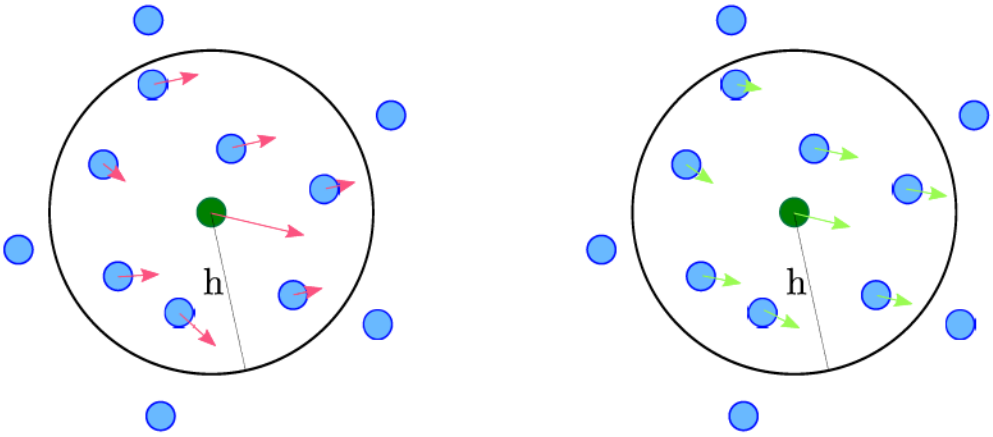
3.1.5 La correction XSPH

Initialement proposée par Monaghan [Mon89], cette technique est plus souvent utilisée afin de modéliser des écoulements à grande vitesse considérés comme turbulents (nombre de Reynolds suffisamment élevé entre 2000 et 3000).

$$\vec{v}_i^* = \vec{v}_i + \alpha_{XSPH} \sum_{j=1}^{n} \frac{m_j}{\rho_j + \rho_i} (\vec{v}_j - \vec{v}_i) W(\vec{r}_i - \vec{r}_j, h) \tag{3.27}$$

Cela a pour effet d'éviter l'inter-pénétration car cette correction permet de maintenir une bonne répartition des particules. La vitesse étant diffusée dans le voisinage de chaque particule, celles-ci conservent effectivement une répartition plus ordonnée (figure 3.4). Cette vitesse corrigée peut être également employée au sein de l'équation de conservation de la masse (3.9). Attention à ne pas utiliser cette technique au sein d'un modèle qui contient une viscosité artificielle ou laminaire sous peine d'ajouter une dissipation trop forte du champ vectoriel des vitesses. Le paramètre

α_{XSPH} définit l'intensité de la correction, il est couramment fixé de façon similaire à α_{AV}. Cette correction rajoute un caractère diffus aux vélocités obtenues ce qui se rapproche de l'interprétation physique de la viscosité. Cette correction est utilisée dans les simulations de [SB12] et [MM13].



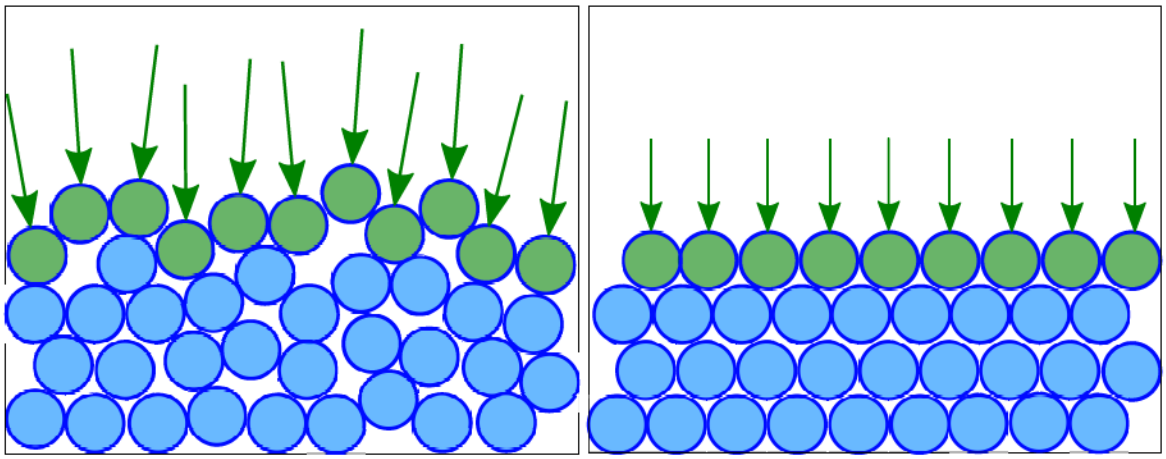
(a) Avant application de la force de viscosité (b) Après application de la force de viscosité

Figure 3.4 – Comportement de la viscosité (correction XSPH)

3.2 Tension de surface

À la surface d'un milieu liquide ou à l'interface entre deux milieux denses, la matière n'est pas, localement, rigoureusement dans le même état. Dans le but de réaliser une simulation dit multiphasique ou simplement à surface libre, il est important de prendre en compte la tension de surface souvent appelée tension superficielle. La tension de surface est symbolisée par une force supplémentaire aux équations de Euler/Navier-Stokes. Cette force est naturellement calculée à partir de la courbure locale de la surface (visible dans la figure 3.5), à l'aide d'une évaluation SPH s'appuyant sur la présence des particules. Cette tension estimée correspond aux forces qui s'exercent à l'interface du fluide. Quand les molécules de fluides sont en surface, elles sont attirées par les molécules au dessous et celles à coté, mais pas par celles extérieures. La résultante est une force dirigée vers l'intérieur du fluide. De cette manière, la cohésion entre molécules, engendre une force tangente à la surface. La

méthode utilisée est basée sur l'article de Müller [MCG03] qui représente un modèle macroscopique de la tension de surface.



(a) Avant application de la force de surface (b) Après application de la force de surface

Figure 3.5 – Comportement de la tension superficielle

Il est important de rappeler que nous souhaitons représenter une force au sein de l'équation (3.2) pour chaque particule :

$$\vec{f}_i^{surface} = \frac{\vec{F}_i^{surface}}{\rho_i}. \tag{3.28}$$

où $\vec{f}_i^{surface}$ représente la force de surface par unité de masse (accélération en $m \cdot s^{-2}$). Dans le but de rechercher la surface du fluide afin d'y appliquer la force de surface, le modèle macroscopique, également appelé communément le modèle de la force de surface continue, est basée sur un champ de couleur c («color field» en anglais). Le champ de couleur à l'image des méthode de réglage de niveau [FF01, LSSF06] («level sets» en anglais), est une fonction définie avec $c = 1$ aux endroits où il y a présence de particules et $c = 0$ ailleurs. Dans la formulation SPH le champ de couleur est définit

pour une particule i de la manière suivante :

$$\begin{aligned} c_i &= c(\vec{r}_i), \\ &= \sum_{j=1}^{n} c_j \frac{m_j}{\rho_j} W_{ij}, \\ &= \sum_{j=1}^{n} \frac{m_j}{\rho_j} W_{ij}. \end{aligned} \tag{3.29}$$

La direction du gradient correspond à celle du vecteur normal à la surface pointant vers l'intérieur du fluide. On obtient à l'aide de (3.30) la normale du fluide, considérée comme la normale $\vec{n}_i$ de chaque particule i.

$$\vec{n}_i = \nabla c_i = \sum_{j=1}^{n} \frac{m_j}{\rho_j} \nabla W_{ij} \tag{3.30}$$

La divergence du gradient autrement dit le Laplacien de ce champ de couleur permet de mesurer la courbure k de la surface :

$$k = \frac{-\nabla \cdot \vec{n}_i}{\|\vec{n}_i\|} = \frac{-\nabla^2 c_i}{\|\vec{n}_i\|}. \tag{3.31}$$

Il est important d'obtenir pour la définition de cette force le signe correct, c'est à dire une courbure positive pour les volumes convexes. Pour cela le signe négatif est introduit au sein de l'équation (3.31). Müller [MCG03] définit la force de surface :

$$\vec{F}_i^{surface} = \varphi k \vec{n}_i = -\varphi \nabla^2 c_i \frac{\vec{n}_i}{\|\vec{n}_i\|}, \tag{3.32}$$

où φ est une constante définie par l'utilisateur. Müller décide de distribuer la force de surface entre les particules à proximité de la surface en multipliant par la normale $\vec{n}_i$. L'évaluation de $\|\vec{n}_i\|$ à des positions où $\|\vec{n}_i\|$ dispose d'une petite valeur causerait des problèmes numériques. Pour cela, Müller calcule la tension de surface si $\|\vec{n}_i\|$ est supérieure à une certaine valeur l, elle aussi définie par l'utilisateur, dans le but d'appliquer la force de surface seulement aux particules qui composent la surface du fluide (figure 3.6).

$$\|\vec{n}_i\| \leq l. \tag{3.33}$$

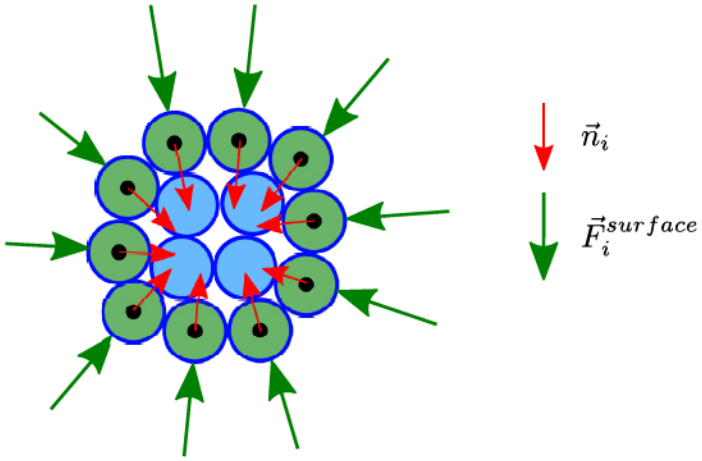


Figure 3.6 – Tension superficielle pour une goutte de fluide

Cependant, il existe dans la littérature [BT07] une autre description des forces de surface à l'aide d'un modèle dit microscopique. Ce modèle se rapproche de l'aspect particulaire du fluide, il considère des forces de cohésion entre les particules et ce, afin d'imiter les forces d'attraction entre les molécules. Contrairement au modèle macroscopique qui se sert du Laplacien afin d'extraire la surface de l'ensemble de particules, le modèle microscopique se sert seulement de la valeur du noyau, car comme nous l'avons vu précédemment l'approximation de la seconde dérivée à l'aide du Laplacien du noyau est susceptible de générer des erreurs. Il est possible d'obtenir une force de surface à l'aide d'une pression artificielle, cette pression est décrite dans le prochain chapitre, plus précisément dans la sous-section 4.2.2.

Chapitre 4

Algorithme de résolution des méthodes particulaires destinée à l'infographie

Dans ce chapitre, nous allons décrire l'ensemble des algorithmes utilisés dans les méthodes particulaires, essentiellement SPH. Dans un premier temps, nous allons résumer l'algorithme dit classique car très largement utilisé. Cette première approche permet de fournir des simulations de qualité au détriment du temps de rendu. Cependant les problèmes de compressibilité du fluide ne sont pas totalement résolus et nécessitent un pas de temps très petit. Ensuite nous verrons les algorithmes dits itératifs qui, à l'aide de sous-étapes de calcul, renforcent l'incompressibilité et permettent de meilleures performances en utilisant des pas de temps significativement plus grands que ceux utilisés dans l'algorithme classique. Comme pour le chapitre 3, avant de lire ce qui suit, il est important de prendre connaissance du contenu de l'annexe B.

4.1 Algorithme dit classique

L'algorithme 4.1 est utilisé dans les simulations [MCG03, BT07]. Il est relativement simple car seulement 4 boucles suffisent à réaliser l'ensemble de la simulation. Les méthodes de recherche de voisinages sont très importantes. Chaque approxima-

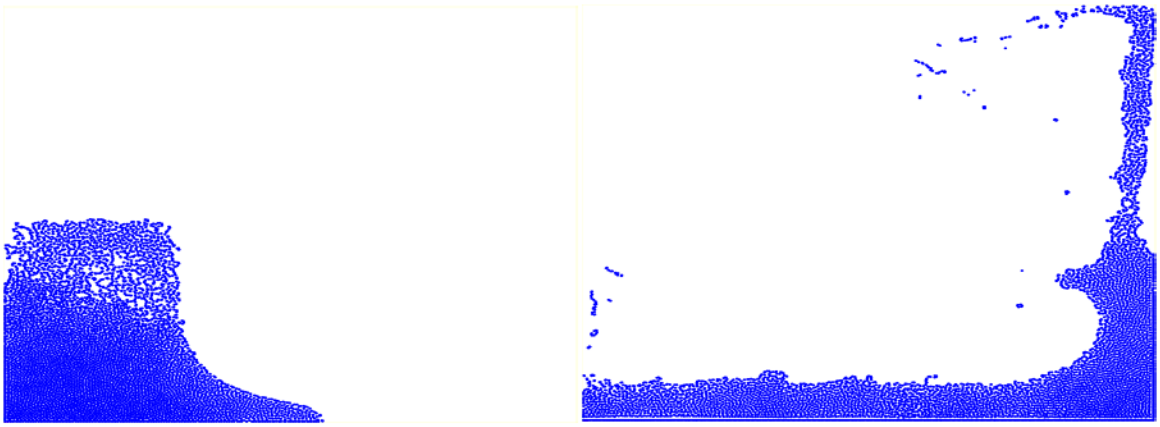
Entrées : Un ensemble de particules i au temps t
Sorties : Un ensemble de particules i au temps $t+1$

pour chaque *particule* i **faire**
  Rechercher le voisinage alors noté $N_i(t)$
fin
pour chaque *particule* i **faire**
  Calculer $\rho_i(t)$ à l'aide de $N_i(t)$
  Calculer $p_i(t)$
fin
pour chaque *particule* i **faire**
  Calculer le gradient de la pression $-\frac{1}{\rho_i(t)}\nabla p_i(t)$ à l'aide de $N_i(t)$
  Calculer les forces extérieures (gravité et surface)
  Calculer la viscosité (artificielle ou non) à l'aide de $N_i(t)$
fin
pour chaque *particule* i **faire**
  Mettre à jour la vitesse $\vec{v}_i(t+1)$
  Mettre à jour la position $\vec{r}_i(t+1)$
  Gérer les cas limites (frontières)
fin

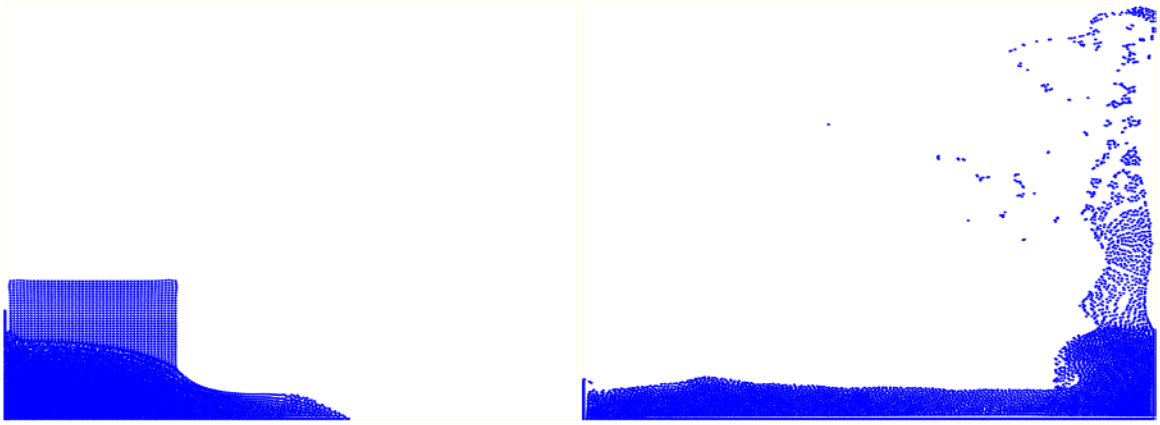
Algorithme 4.1 : Boucle de simulation SPH/WCSPH

tion SPH nécessite de connaître le voisinage de particules $N_i(t)$ situées à une distance euclidienne inférieure ou égale au support du noyau (h ou $2h$ suivant la fonction noyau employée). Le fait de connaître le voisinage au préalable nous évite d'itérer doublement sur l'ensemble des particules. Différentes méthodes d'optimisation de recherche de voisinage existent, ces méthodes sont décrites dans la sous-section 6.3.3.

L'obtention de la densité peut se faire à l'aide de la sommation SPH (3.4) mais elle peut également se faire à l'aide de l'équation de conservation de la masse (3.9). La figure 4.1 met en perspective la différence de compressibilité entre une simulation SPH utilisant la sommation SPH et l'équation des gaz parfaits (3.13) pour l'obtention de p_i et une simulation de type WCSPH utilisant l'équation de conservation de la masse et l'équation des Tait (3.14). Toutefois une approche de type faiblement compressible nécessite un plus petit pas de temps afin de respecter les critères de stabilité de type CFL décrits dans l'annexe B.



(a) Simulation basée sur [MCG03] de type SPH fortement compressible avec $\Delta t = 0.0001$



(b) Simulation basée sur [BT07] de type WCSPH faiblement compressible avec $\Delta t = 0.00001$

Figure 4.1 – Illustrations à différents temps t des simulations SPH et WCSPH

L'ensemble des forces calculées permet de mettre à jour l'accélération selon l'équation (3.2) et ainsi la position, la vitesse de chaque particule. Les méthodes d'intégrations numériques sont décrites dans l'annexe B, plus précisément dans la section B.2. Il convient de judicieusement choisir le schéma numérique d'intégration pour obtenir nos résultats, la vitesse pour l'équation de conservation du mouvement ou encore la densité pour la conservation de la masse et ce afin de conserver un bon compromis entre la simulation et les performances de rendu.

4.2 Algorithme dit itératif

4.2.1 Simulation SPH incompressible, prédite et corrective

«Predictive-Corrective Incompressible SPH» (PCISPH) en anglais [SP09] est un des premiers algorithmes de résolution des équations de la dynamique des fluides de manière itérative. Celui-ci permet de résoudre les problèmes d'incompressibilité et de disposer d'un pas de temps raisonnable afin de fournir une simulation de fluide efficace. Pour mieux le comprendre nous allons en décrire les étapes clés.

On note $\rho^*(t+1)$ la densité intermédiaire prédite, la densité obtenue à l'aide des valeurs de postions intermédiaires $\vec{r}^*$:

$$\rho_i^*(t+1) = \sum_{j=1}^{n} m_j W(\vec{r}_i^*(t+1) - \vec{r}_j^*(t+1), h), \tag{4.1}$$

$$= \sum_{j=1}^{n} m_j W(\vec{r}_i^*(t) + \Delta\vec{r}_i^*(t) - \vec{r}_j^*(t) - \Delta\vec{r}_j^*(t), h), \tag{4.2}$$

$$= \sum_{j=1}^{n} m_j W(\vec{r}_{ij}^*(t) + \Delta\vec{r}_{ij}^*(t), h). \tag{4.3}$$

En appliquant l'approximation de Taylor du premier ordre et en considérant $\Delta\vec{r}_{ij}^*(t)$ suffisamment petit. On a :

$$\rho_i^*(t+1) = \rho_i^*(t) + \Delta\rho_i^*(t). \tag{4.4}$$

Par la suite on s'intéresse à la force de pression, en utilisant le gradient symétrique (3.19). Les auteurs, Solenthaler et Pajarola [SP09] calculent la pression de la manière suivante (4.5). La pression est obtenue en dérivant l'expression de la force de pression afin d'obtenir une pression en adéquation avec la densité cible ρ_0. On rajoute à la pression $p_i(t)$, une fonction de $\rho_{err_i}^*(t+1)$ susceptible de corriger la pression au sein du calcul de la ligne 20 de l'algorithme 4.2. L'équation utilisée (4.7) se rapproche de

l'équation d'état des gaz parfaits (3.13) où $k = 1$.

$$p_i(t) = p_i(t) + f\left(\rho^*_{err_i}(t+1)\right), \tag{4.5}$$

$$f\left(\rho^*_{err_i}(t+1)\right) = \delta\rho^*_{err_i}(t+1), \tag{4.6}$$

$$\rho^*_{err_i}(t+1) = \rho^*_i(t+1) - \rho_0, \tag{4.7}$$

avec δ qui vaut :

$$\delta = -\frac{\rho_0^2}{2\Delta t^2 m_i^2 \left(-\sum_{j=1}^n \nabla W_{ij} \cdot \sum_{j=1}^n \nabla W_{ij} - \sum_{j=1}^n (\nabla W_{ij} \cdot \nabla W_{ij})\right)}. \tag{4.8}$$

Pour plus de détails sur l'obtention de δ, se référer directement à l'article [SP09]. On retrouve l'ensemble des calculs dans l'algorithme 4.2. La particularité de cet algorithme est que le calcul de la pression se fait de manière additive. De cette façon, on obtient une force de pression corrigée afin de permettre à $\rho_i(t+1)$ de converger vers la densité au repos du fluide ρ_0 et donc de renforcer l'incompressibilité. Nous décidons de prédire l'évolution de la particule pour en déduire sa densité et ainsi la corriger en itérant sur la force de pression. Ceci permet d'obtenir des forces suffisamment faibles et donc des accélérations respectant plus facilement les conditions CFL, nous permettant d'augmenter sensiblement le pas de temps.

Cet algorithme dispose d'une méthode qui itère jusqu'à un certain nombre d'étapes $IterationMinimum$ défini par l'utilisateur ($1 \leq IterationMinimum \leq 50$). Cette méthode itère également jusqu'à ce que l'erreur globale de densité (4.9) au sein du fluide soit inférieure à un certain seuil η défini lui aussi par l'utilisateur (1% ou 10% par exemple).

$$\rho_{err}(t+1) = \sum_{j=1}^{n} \frac{\rho^*_{err_i}(t+1)}{n}. \tag{4.9}$$

Entrées : Un ensemble de particules i au temps t
Sorties : Un ensemble de particules i au temps $t+1$

pour chaque *particule* i **faire**
  Rechercher le voisinage alors noté $N_i(t)$
fin
pour chaque *particule* i **faire**
  Calculer $\rho_i(t)$
  Calculer la viscosité (artificielle ou non) et les forces de gravité
  Initialiser $p_i(t) = 0$
fin
tant que $\rho_{err}(t+1) > \eta \parallel iter < IterationMinimum$ **faire**
  pour chaque *particule* i **faire**
    Prédire la vitesse $\vec{v}_i^*(t+1)$
    Prédire la position $\vec{r}_i^*(t+1)$
  fin
  pour chaque *particule* i **faire**
    Prédire la densité $\rho_i^*(t+1)$ à l'aide de $\vec{r}_i^*(t+1)$
    Calculer l'erreur de densité $\rho^*_{err_i}(t+1)$
    Mettre à jour la pression $p_i(t) = p_i(t) + f\left(\rho^*_{err_i}(t+1)\right)$
  fin
  pour chaque *particule* i **faire**
    Calculer le gradient de la pression $-\frac{1}{\rho_i(t)^*}\nabla p_i(t)$
  fin
  Calculer $\rho_{err}(t+1)$
  $iter = iter + 1$
fin
pour chaque *particule* i **faire**
  Mettre à jour la vitesse $\vec{v}_i(t+1)$ et la position $\vec{r}_i(t+1)$
  Gérer les cas limites (frontières)
fin

Algorithme 4.2 : Boucle de simulation PCISPH

Cependant, cet algorithme comporte un désavantage. Il se situe dans la recherche de voisinage. En effet la densité prédite $\rho_i^*(t+1)$, est calculée à l'aide du voisinage défini durant l'étape t, $N_i(t)$, ce qui augmente le taux d'erreur des estimations de la densité prédite et ainsi de la pression. Pour des raisons d'efficacité, les auteurs, Solenthaler et Pajarola réutilisent le même voisinage.

4.2.2 Simulation de fluide basée sur la position

La simulation de fluide basée sur la position («Position Based Fluids» en anglais) [MM13] est un moyen de simuler des liquides, suite aux écrits de Müller sur la dynamique basée sur la position [MHHR07] («Position Based Dynamics» en anglais). Position Based Dynamics (PBD) dispose d'algorithmes et de principes qui sont utilisés pour la simulation de vêtements déformables dans le SDK («Software Development Kit» en anglais) PhysX de Nvidia. Sur le principe itératif de PBD, la simulation de fluide basée sur la position (PBF) peut maintenir l'incompressibilité plus efficacement que l'algorithme classique SPH. L'algorithme 4.3 est très similaire à celui décrit précédemment. Il a également un terme de pression artificielle qui améliore la distribution des particules et crée des effets de tension, permettant une qualité de surface de fluide supérieure.

Miles Macklin [MHHR07] considère la même masse m_i pour chaque particule et définit une contrainte de densité sur la position de chaque particule de la manière suivante :

$$C_i(\vec{r}_1, \ldots, \vec{r}_n) = \frac{\rho_i}{\rho_0} - 1. \tag{4.10}$$

À l'image de l'équation de Tait (3.14) où $B = 1$ et $\gamma = 1$, cette contrainte nous fournit une pseudo-pression relative. Par la suite, cette contrainte doit respecter l'équation suivante (4.11). Peu importe la variation des positions des particules i, la conservation de la masse, en l'occurrence la variation de la densité de chaque particule, doit être constante :

$$C_i(\vec{r} + \Delta\vec{r}) = 0. \tag{4.11}$$

La méthode de Newton-Raphson permet de résoudre les équations de la forme $f(x) = 0$ en trouvant les valeurs de x pour lesquelles l'égalité est valable. En appliquant la méthode de Newton-Raphson, les auteurs Macklin et Müller décrivent les relations suivantes :

$$\Delta\vec{r} \approx \nabla C(\vec{r})\lambda, \tag{4.12}$$

$$C_i\left(\vec{r} + \Delta\vec{r}\right) \approx C(\vec{r}) + \nabla C \cdot \Delta\vec{r} = 0, \tag{4.13}$$

$$\approx C(\vec{r}) + \nabla C \cdot \nabla C(\vec{r})\lambda = 0. \tag{4.14}$$

Entrées : Un ensemble de particules i au temps t
Sorties : Un ensemble de particules i au temps $t+1$

pour chaque *particule* i **faire**
  Calculer la vitesse intermédiaire $\vec{v}_i^*(t)$ (avec la force de gravité) et la position intermédiaire $\vec{r}_i^*(t)$
fin
pour chaque *particule* i **faire**
  Rechercher le voisinage alors noté $N_i^*(t)$ à l'aide de $\vec{r}_i^*(t)$
fin
tant que $iter < IterationMinimum$ **faire**
  pour chaque *particule* i **faire**
    Calculer la densité intermédiaire $\rho_i^*(t)$
    Calculer λ_i à l'aide de $\vec{r}_i^*(t)$ et $\rho_i^*(t)$
  fin
  pour chaque *particule* i **faire**
    Calculer la variation de position $\Delta\vec{r}_i^*$
    Gérer les cas limites (frontières)
  fin
  pour chaque *particule* i **faire**
    Prédire la prochaine position $\vec{r}_i^*(t+1) = \vec{r}_i^*(t) + \Delta\vec{r}_i^*$
  fin
  $iter = iter + 1$
fin
pour chaque *particule* i **faire**
  Mettre à jour la vitesse $\vec{v}_i(t+1) = \frac{1}{\Delta t}\left(\vec{r}_i^*(t+1) - \vec{r}_i(t)\right)$
  Appliquer la correction XSPH
  Mettre à jour la position $\vec{r}_i(t+1) = \vec{r}_i^*(t+1)$
fin

Algorithme 4.3 : Boucle de simulation de fluide basée sur la position

En utilisant la sommation SPH sur le gradient d'une fonction on obtient le gradient de la contrainte (4.11) pour une particule k :

$$\nabla_{\vec{r}_k} C_i = \frac{1}{\rho_0} \sum_{j=1}^{n} \nabla_{\vec{r}_k} W\left(\vec{r}_i - \vec{r}_j, h\right). \tag{4.15}$$

Cette équation nous fournit deux possibilités, en nous appuyant sur la symétrie du gradient du noyau SPH :

$$\nabla_{\vec{r}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_{j=1}^{n} \nabla_{\vec{r}_k} W\left(\vec{r}_i - \vec{r}_j, h\right) & \text{si } k = i, \\ -\nabla_{\vec{r}_k} W\left(\vec{r}_i - \vec{r}_j, h\right) & \text{si } k = j. \end{cases} \tag{4.16}$$

En intégrant les résultats précédents dans l'équation (4.14), on peut donc très facilement obtenir λ_i :

$$\lambda_i = -\frac{C_i(\vec{r}_1, \ldots, \vec{r}_n)}{\sum_k \|\nabla_{\vec{r}_k} C_i\|^2}, \tag{4.17}$$

et définir la variation de position qui satisfait l'équation (4.11) :

$$\Delta\vec{r}_i = \frac{1}{\rho_0} \sum_{j=1}^{n} (\lambda_i + \lambda_j) \nabla W(\vec{r}_i - \vec{r}_j, h). \tag{4.18}$$

Cette dernière équation (4.18) peut paraître complexe, cependant il s'agit seulement d'estimer une variation de position de la même manière que le calcul de force de pression (en utilisant le gradient symétrique (3.19)) :

$$\Delta\vec{r}_i = \sum_{j=1}^{n} m_j \left(\frac{p_j}{\beta_j} + \frac{p_i}{\beta_i} \right) \nabla W(\vec{r}_i - \vec{r}_j, h), \tag{4.19}$$

où β_j et β_i représentent respectivement une fonction de λ_i et λ_j. Cette méthode itère de façon similaire à l'algorithme 4.2 sauf pour le calcul de l'erreur globale de densité. L'utilisateur définit le nombre d'itérations minimum en fonction de la simulation, turbulente ou non. Cependant PBF fournit une solution au principal problème de recherche de voisinage de PCISPH. Il calcul le voisinage à l'aide d'une position intermédiaire $\vec{r}_i^*$.

La pression artificielle utilisée est celle de Monaghan [Mon00], qui permet de résoudre les problèmes de l'estimation du gradient dans le cas où un nombre insuffisant de particules sont proches. Clavet [CBP05] utilise une seconde pression «proche» pour conserver une bonne cohésion des particules tout en facilitant l'estimation du gradient de la pression. Afin d'éviter les valeurs négatives de la pression, Miles Macklin rajoute une pression artificielle au sein du calcul de la variation de position :

$$\Delta\vec{r}_i = \frac{1}{\rho_0} \sum_{j=1}^{n} (\lambda_i + \lambda_j + p_{artificielle}) \nabla W(\vec{r}_i - \vec{r}_j, h), \tag{4.20}$$

où $p_{artificielle}$ est :

$$p_{artificielle} = -\tau \left(\frac{W(\vec{r}_i - \vec{r}_j, h)}{W(\epsilon h, h)} \right)^{\sigma}. \tag{4.21}$$

Au sein de nos simulations (visibles sur la figure 4.2) en deux dimensions, nous avons $\tau = 0.0000008$, $\epsilon = 0.2$ et $\sigma = 4$ qui semble nous fournir de très bons résultats. Les auteurs fournissent d'autres constantes à utiliser en trois dimensions.

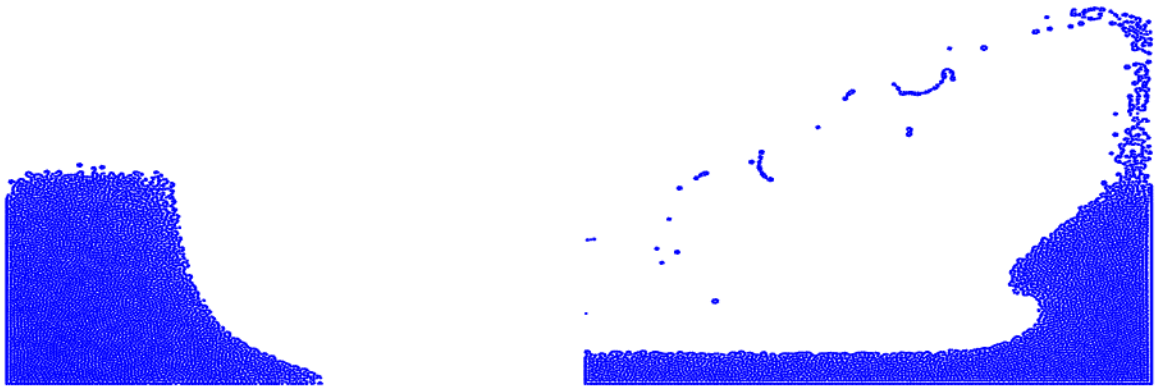


Figure 4.2 – Illustrations à différents temps t des simulations PBF avec $\Delta t = 0.01$

Cette approche nous permet d'obtenir une simulation très faiblement compressible voire incompressible avec un pas de temps relativement grand (figure 4.2) pour des résultats similaire à la figure 4.1.

4.3 Rendement et discussion

4.3.1 Rendement

Le rendement des algorithmes itératifs est généralement caractérisé par le pas de temps maximal possible et le nombre d'itérations requis pour une erreur de densité spécifiée. Solenthaler et Pajarola [SP09] montrent que PCISPH permet d'obtenir un pas de temps qui est jusqu'à deux fois plus grand que celui utilisé au sein de simulations SPH faiblement compressibles [BT07] (algorithme classique). Le nombre moyen d'itérations est compris entre trois et cinq pour des erreurs globales de densité comprises en 1% et 10% pour des simulations de type bris de barrage (figure 4.1-figure 4.2-figure 6.8). PBF tolère des pas de temps plus importants, cependant cet algorithme nécessite plus d'itérations, résultant en une performance globale similaire. Toutefois, comme indiqué dans [MM13] une analyse approfondie des performances des

algorithmes itératifs est plutôt compliquée. Cela est dû au fait que ces algorithmes n'atteignent pas nécessairement leur performance optimale pour le plus grand pas de temps possible. D'une part, le nombre d'itérations augmente avec le pas de temps. D'autre part, la recherche de voisinage rentre en ligne de compte.

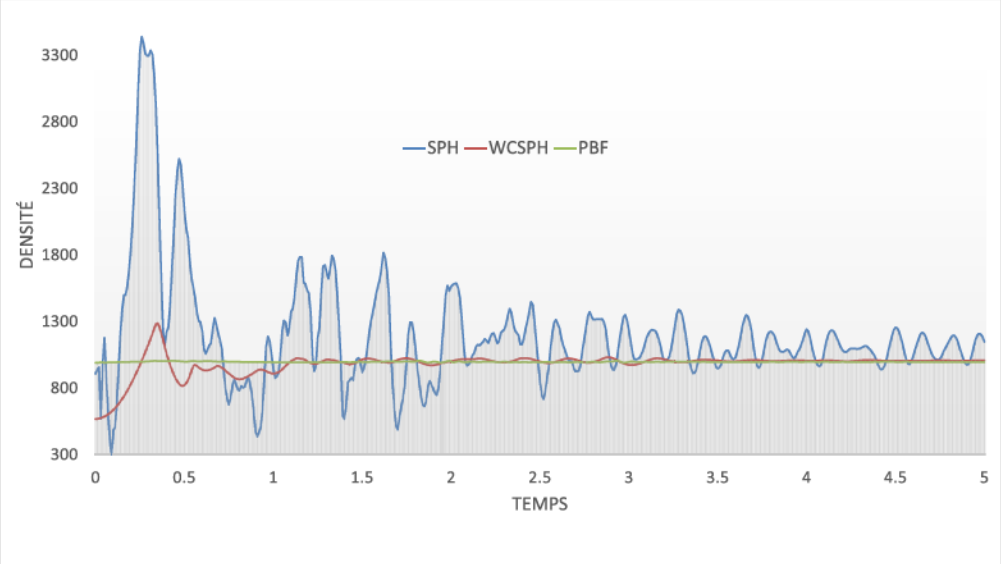


Figure 4.3 – Comparaison de la moyenne des densités par rapport aux temps

La figure 4.3 permet de comparer graphiquement la moyenne des densités pour les simulations visibles dans la figure 4.2 et la figure 4.1, avec un ensemble de 5000 particules en deux dimensions. On constate l'importance de l'usage d'algorithmes itératifs au sein de nos simulations. Les implantations que nous avons des algorithmes itératifs de ces dernières années permettent bien de réaliser des simulations de fluides dits incompressibles.

4.3.2 Discussion

Les algorithmes dit itératifs utilisent des équations d'état et des constantes de rigidité. Ils diffèrent également en fonction de nombreuses constantes. Il serait intéressant d'analyser ces aspects : quelle constante de rigidité est optimale ? Quelles sont les constantes à utiliser ? Quel est le domaine (en taille et unité) ? Peut-on réel-

lement définir des constantes proches de la physique ? Un questionnement auquel très peu d'articles peuvent répondre par manque d'explication du raisonnement ou par manque de détails quant aux constantes utilisées. Il serait intéressant de réaliser une analyse des constantes optimales.

Chapitre 5

Animation lagrangienne à l'aide des noyaux constants par morceaux

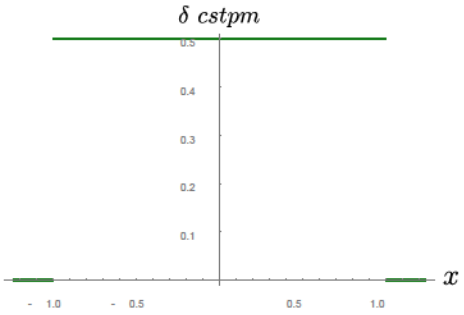
Au sein de ce chapitre, nous proposons d'apporter une utilisation des noyaux constants par morceaux, développés par Jean-Marc Belley, Philippe Belley, Fabrice Colin et Richard Egli [BBCE09]. Après une description en une dimension de ces noyaux et de leurs usages [BBCE09], nous appliquons ces noyaux en deux dimensions dans une approche lagrangienne en comparaison avec les méthodes SPH classiques. La méthode est basée sur une résolution matricielle [CESM11] d'un système linéaire, nous utilisons également une approche de programmation linéaire, permettant de fournir des résultats encourageants pour une éventuelle utilisation dans des méthodes eulériennes.

5.1 Définition des noyaux constants par morceaux en une dimension

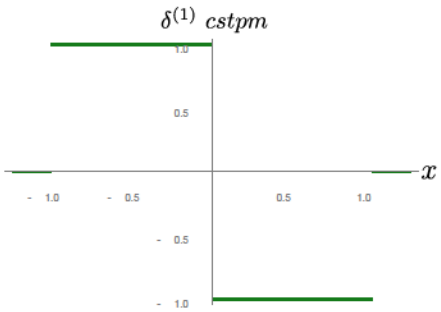
Dans ce mémoire nous utiliserons trois noyaux discontinus (figure 5.1) précédemment utilisés par [BBCE09, CESM11, Sam14]. Ces noyaux seront employés pour estimer une fonction de même que ses premières et secondes dérivées. L'idée à l'origine de la notion des noyaux constants par morceaux est la même que pour les noyaux

SPH. En effet, ces noyaux décrits plus loin, respectent les mêmes conditions que les noyaux SPH au sens des distributions. Comme les noyaux SPH, ils obéissent à certaines conditions de reproduction des fonctions polynomiales jusqu'à un degré donné.

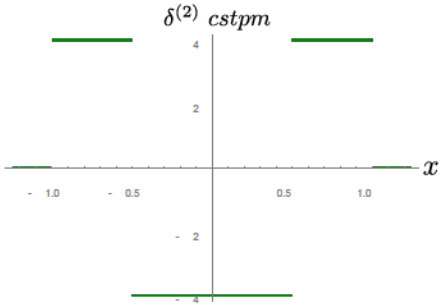
Pour simplifier la compréhension, nous utilisons la notation précédemment introduite dans [CESM11, Sam14]. Disposant d'un support de noyau h, le premier noyau (figure 5.1a), que nous dénoterons $\delta_{cstpm}(x,h)$ pour $h>0$, est défini par :



(a) noyau constant par morceaux (pour estimer la fonction)



(b) la formulation du noyau pour estimer la dérivée première



(c) la formulation du noyau pour estimer la dérivée seconde

Figure 5.1 – Graphe du noyau constant par morceaux, $\delta_{cstpm}(x,h)$ pour $h=1$

$$\delta_{cstpm}(x,h)=\begin{cases}\frac{1}{2h} & \text{si} \quad -h\leq x\leq h,\\ 0 & \text{sinon.}\end{cases} \tag{5.1}$$

Le deuxième noyau (figure 5.1b) représente la première dérivée au sens des dérivées faibles du noyau $\delta_{cstpm}(x,h)$. Pour cela il est noté $\delta^{(1)}_{cstpm}$ et est défini par :

$$\delta^{(1)}_{cstpm}(x,h) = \begin{cases} \frac{1}{h^2} & \text{si} \quad -h \leq x \leq 0, \\ -\frac{1}{h^2} & \text{si} \quad 0 \leq x \leq h, \\ 0 & \text{sinon.} \end{cases} \tag{5.2}$$

Et le dernier noyau (figure 5.1c), noté $\delta^{(2)}_{cstpm}$:

$$\delta^{(2)}_{cstpm}(x,h) = \begin{cases} \frac{4}{h^3} & \text{si} \quad -\frac{h}{2} \leq x \leq h, \\ -\frac{4}{h^3} & \text{si} \quad 0 \leq x \leq \frac{h}{2}, \\ 0 & \text{sinon.} \end{cases} \tag{5.3}$$

Ils seront utilisés, respectivement, pour estimer la fonction f et ses deux premières dérivées en un point, en l'occurrence la position d'une particule. À la différence des noyaux SPH où la dérivation directe du noyau permet d'obtenir un noyau pour l'approximation des dérivées d'une fonction, on définit ici un noyau pour chacune des dérivées. Leur usage en une dimension n'est plus à démontrer, dans [BBCE09], les auteurs décrivent la précision de l'estimation des noyaux CSTPM (constants par morceaux) et prouvent que justement ces noyaux sont plus précis comparés aux résultats obtenus avec le noyau «Poly6».

Nous soulignerons au sein de ce chapitre, la capacité des noyaux au décentrage ainsi que leur utilisation dans le cas d'une simulation lagrangienne.

5.2 Définition des noyaux constants par morceaux en deux dimensions

5.2.1 Définition des noyaux constants par morceaux en deux dimensions

Les noyaux précédents permettent d'effectuer des approximations selon une seule dimension. Dans [CESM11], Colin, Egli et Serghini généralisent à plusieurs dimensions

l'usage de ces noyaux. Il suffit d'effectuer le produit de deux noyaux en une dimension pour obtenir le noyau souhaité en deux dimensions. Soit $\vec{r} = [x, y]^T$, on souhaite estimer la fonction et ses dérivées en x et en y jusqu'au second ordre.

$$\begin{aligned} W_{cstpm} &:= \delta_{cstpm}(x)\delta_{cstpm}(y), \\ \partial W_{cstpm}/\partial x &:= \delta^{(1)}_{cstpm}(x)\delta_{cstpm}(y), \\ \partial W_{cstpm}/\partial y &:= \delta^{(1)}_{cstpm}(y)\delta_{cstpm}(x), \\ \partial^2 W_{cstpm}/\partial x^2 &:= \delta^{(2)}_{cstpm}(x)\delta_{cstpm}(y), \\ \partial^2 W_{cstpm}/\partial y^2 &:= \delta^{(2)}_{cstpm}(y)\delta_{cstpm}(x), \end{aligned}$$

avec $\partial W_{cstpm}/\partial x$ la dérivée première en x, $\partial W_{cstpm}/\partial y$ la dérivée première en y ainsi que $\partial^2 W_{cstpm}/\partial x^2$ la dérivée seconde en x et $\partial^2 W_{cstpm}/\partial y^2$ la dérivée seconde en y. En utilisant les conditions d'interpolations et le développement en série de Taylor en plusieurs dimensions, Colin, Egli et Serghini [CESM11] montrent que les ordres de convergence sont de l'ordre de $O(h)$ pour la première dérivée. L'ordre de convergence est l'erreur relative décrite dans le chapitre 2. Ces noyaux sont plus précis que les noyaux SPH au moment de l'évaluation numérique des intégrales des produits de convolution.

5.2.2 Usage des noyaux constants par morceaux en deux dimensions

À partir du principe des méthodes particulaires et des conditions d'interpolations des noyaux, on peut, pour une particule i sur un voisinage j de n particules, évaluer la dérivée k-ième de f à l'aide des noyaux CSTPM (constants par morceaux) en deux dimensions. Colin, Egli et Serghini [CESM11] proposent une méthode évaluant des poids correspondant à chaque noyau au voisinage d'un point :

$$\langle\, f^{(k)}(\vec{r}_i)\,\rangle \approx \int_\Omega f(\vec{r}_j) W^{(k)}_{i,j} d\Omega \approx \sum_{j=1}^{n} f(\vec{r}_j) w^{(k)}_{i,j} \tag{5.4}$$

où $w^{(k)}_{i,j}$ représente les poids qui permettent d'évaluer la dérivée k-ième, et $W^{(k)}_{i,j}$ représente la dérivée k-ième du noyau.

Pour cela, on construit les matrices de résolution $\mathbf{M}$ de la façon suivante :

$$\mathbf{M}^{(0,1)} = \begin{bmatrix} 1 & 1 & ... & 1 \\ x_1 - x_i & x_2 - x_i & ... & x_n - x_i \\ y_1 - y_i & y_2 - y_i & ... & y_n - y_i \end{bmatrix} ; \tag{5.5}$$

$$\mathbf{M}^{(2)} = \begin{bmatrix} 1 & 1 & ... & 1 \\ x_1 - x_i & x_2 - x_i & ... & x_n - x_i \\ y_1 - y_i & y_2 - y_i & ... & y_n - y_i \\ (x_1 - x_i)(y_1 - y_i) & (x_2 - x_i)(y_2 - y_i) & ... & (x_n - x_i)(y_n - y_i) \\ (x_1 - x_i)^2 & (x_2 - x_i)^2 & ... & (x_n - x_i)^2 \\ (y_1 - y_i)^2 & (y_2 - y_i)^2 & ... & (y_n - y_i)^2 \end{bmatrix} . \tag{5.6}$$

Avec $\mathbf{M}^{(0,1)}$ la matrice $\mathbf{M}$ utilisée pour estimer la fonction ainsi que sa première dérivée en x_i et en y_i. $\mathbf{M}^{(2)}$ est la matrice utilisée pour estimer la seconde dérivée en x_i et en y_i. Le but est d'obtenir l'ensemble des poids $w_{i,j}^{(k)}$ qui vont nous permettre d'estimer la fonction et ses dérivées k-ième. On notera donc W_{cstpm} le vecteur contenant les poids pour l'estimation de la fonction, $\partial W_{cstpm}/\partial x$ pour l'estimation de la première dérivée en x_i de la fonction et $\partial^2 W_{cstpm}/\partial x^2$ pour l'estimation de la seconde dérivée en x_i :

$$W_{cstpm} = \begin{bmatrix} w_{i,1}^{(0)} \\ w_{i,2}^{(0)} \\ ... \\ w_{i,n}^{(0)} \end{bmatrix} ; \tag{5.7 a}$$

$$\partial W_{cstpm}/\partial x = \begin{bmatrix} w_{i,1}^{(1)} \\ w_{i,2}^{(1)} \\ ... \\ w_{i,n}^{(1)} \end{bmatrix} ; \qquad \partial W_{cstpm}/\partial y = \begin{bmatrix} w_{i,1}^{(1)} \\ w_{i,2}^{(1)} \\ ... \\ w_{i,n}^{(1)} \end{bmatrix} ; \tag{5.7 b,c}$$

$$\partial^2 W_{cstpm}/\partial x^2 = \begin{bmatrix} w_{i,1}^{(2)} \\ w_{i,2}^{(2)} \\ ... \\ w_{i,n}^{(2)} \end{bmatrix} ; \qquad \partial^2 W_{cstpm}/\partial y^2 = \begin{bmatrix} w_{i,1}^{(2)} \\ w_{i,2}^{(2)} \\ ... \\ w_{i,n}^{(2)} \end{bmatrix} . \tag{5.7 d,e}$$

Nous cherchons à résoudre un système de type $\mathbf{Ax} = \mathbf{b}$ où $\mathbf{A}$ représente la matrice $\mathbf{M}$ (5.5) (5.6) et $\mathbf{x}$ représente les poids définis dans les équations (5.7). Le vecteur $\mathbf{b}$ permet de satisfaire les conditions d'interpolation du noyau afin justement d'obtenir l'estimation adéquate de la fonction et de ses dérivées :

$$\mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ pour } W_{cstpm} \text{ avec } \mathbf{M}^{(0,1)} ; \tag{5.8}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ pour } \frac{\partial W_{cstpm}}{\partial x} \text{ avec } \mathbf{M}^{(0,1)} ; \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ pour } \frac{\partial W_{cstpm}}{\partial y} \text{ avec } \mathbf{M}^{(0,1)} ; \tag{5.9}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 0 \end{bmatrix} \text{ pour } \frac{\partial^2 W_{cstpm}}{\partial x^2} \text{ avec } \mathbf{M}^{(2)} ; \qquad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \end{bmatrix} \text{ pour } \frac{\partial^2 W_{cstpm}}{\partial y^2} \text{ avec } \mathbf{M}^{(2)}. \tag{5.10}$$

Pour plus d'informations sur la constitution des vecteurs $\mathbf{b}$ afin de satisfaire les conditions d'interpolation et de représentation des fonctions polynomiales, nous invitons le lecteur à se référer à [CESM11].

La résolution de ces systèmes permet donc d'attribuer des poids à l'ensemble des particules présentes dans le voisinage d'une particule pour chaque dimension et pour les dérivées première et seconde. Il faut souligner que d'autres conditions d'ordre pratique sont nécessaires afin de résoudre un tel système. Le nombre de particules à l'intérieur du voisinage, en deux dimensions, doit être de trois au minimum pour la première dérivée et de six pour la seconde. Si le nombre de particules à l'intérieur de h est trop faible, le système ne possède pas de solutions. Évidemment, il faut faire en sorte que le positionnement des particules à l'intérieur du noyau permette la résolution du système. Il faut s'assurer que les particules ne soient pas toutes alignées

selon l'axe x ou y. Il faut cependant noter que la précision du noyau dépend de la distance de la particule la plus éloignée $h_{réel}$ et non du support $h_{théorique}$ visible dans la figure 5.2.

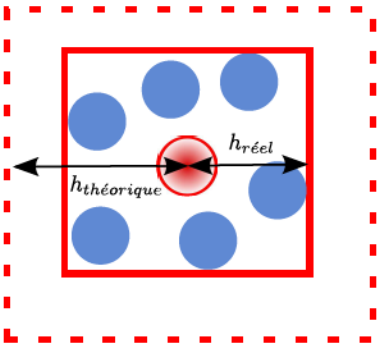


Figure 5.2 – Supports des noyaux CSTPM

5.2.3 Comparaison avec la méthode SPH

Rappels sur l'approximation SPH

A partir du principe de l'estimation SPH, on peut approximer à l'aide d'une somme finie la valeur de f en $\vec{r}_i$ et de ses dérivées. f peut être un scalaire ou un vecteur et est définie pour des particules à une position $\vec{r}_j$ en deux ou trois dimensions dimensions.

$$\langle\ f^{(k)}(\vec{r}_i)\ \rangle \approx \int_\Omega f(\vec{r}_j) W^{(k)}_{sph}(\vec{r}_i - \vec{r}_j, h) d\Omega \approx \sum_j f(\vec{r}_j) W^{(k)}_{ij} V_j \tag{5.11}$$

V_j est le volume occupé par la particule j. Selon la méthode SPH son volume est donné par $V_j = \frac{m_j}{\rho_j}$ où m_j est la masse associée à la particule j et ρ_j est la densité de celle-ci. h représente le support du noyau.

Différence entre les approximations

Opérateur	Définition	Approximation SPH	Approximation Noyaux discontinus
Gradient	$\nabla p = \begin{bmatrix} \frac{\partial p}{\partial x} \\ \frac{\partial p}{\partial y} \end{bmatrix}$	$\nabla f_i = \sum_{j=1}^{n} m_j \left(\frac{f_j}{\rho_j}\right) \nabla W_{sph}(\vec{r}_i - \vec{r}_j, h)$	$\nabla f_i = \sum_{j=1}^{n} f_j \nabla W_{cstpm}$
Divergence	$\nabla \cdot \vec{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}$	$\nabla \cdot \vec{f}_i = \sum_{j=1}^{n} m_j \left(\frac{\vec{f}_j}{\rho_j}\right) \cdot \nabla W_{sph}(\vec{r}_i - \vec{r}_j, h)$	$\nabla \cdot \vec{f}_i = \sum_{j=1}^{n} \vec{f}_j \cdot \nabla W_{cstpm}$
Laplacien	$\nabla^2 \vec{v} = \frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_y}{\partial y^2}$	$\nabla^2 \vec{f}_i = \sum_{j=1}^{n} m_j \left(\frac{\vec{f}_j}{\rho_j}\right) \nabla^2 W_{sph}(\vec{r}_i - \vec{r}_j, h)$	$\nabla^2 \vec{f}_i = \sum_{j=1}^{n} \vec{f}_j \nabla^2 W_{cstpm}$

Tableau 5.1 – Tableau résumant les deux types d'approximations

f_i est la valeur de la fonction f pour la particule i : $f(\vec{r}_i)$. W_{sph} n'est autre qu'un noyau SPH, ∇W_{sph} son gradient et $\nabla^2 W_{sph}$ son Laplacien. ∇W_{cstpm} représente l'ensemble des poids calculés $\partial W_{cstpm}/\partial x$ et $\partial W_{cstpm}/\partial y$ à l'aide des noyaux constants par morceaux afin d'estimer la dérivée première de la fonction f en x et y. Il est important de noter l'absence du terme d'unité de volume $V_j = \frac{m_j}{\rho_j}$ dans le cas d'une approximation à l'aide des noyaux constants par morceaux. Celui-ci se retrouve implicitement inclus au sein de la valeur des poids obtenue pour chaque particule j du voisinage.

Pour comprendre comment cette valeur se retrouve implicite au sein du calcul de l'approximation avec les nouveaux noyaux, nous allons nous inspirer de la méthode de quadrature de Gauss. Dans un cas unidimensionnel, on a :

$$\langle\ f(x)\ \rangle = \lim_{h \to 0} \int_{\Omega} f(x') W(x - x', h) dx', \tag{5.12}$$

avec dx' qui représente en SPH notre unité de volume V_j. Pour l'usage des noyaux constants par morceaux on a :

$$\{f(x)\}_i \approx \sum_j f_j W_{ij} w'_{ij}, \tag{5.13}$$

avec W_{ij} qui représente la valeur du noyau présente dans la définition (5.1) et w'_{ij} qui représente notre pas d'intégration, pour cela lors de la conception de nos matrices $\mathbf{M}$ il est nécessaire d'y inscrire les différences de position. Par la suite, on pose :

$$w_{ij} := W_{ij} w'_{ij}. \tag{5.14}$$

On peut donc se retrouver avec des valeurs de poids $w_{ij} < 0$ car w'_{ij}, qui peut être négatif, se retrouvent implicitement inclus durant le calcul des poids.

5.3 Usage des noyaux constants par morceaux en deux dimensions

Au sein de cette section, nous décrivons comment nous avons appliqué la méthode des noyaux constants par morceaux afin de résoudre les équations de la dynamique des fluides. Il est important de prendre connaissance de la section 3.1 précédente, pour aider à la compréhension de la résolution des équations de Euler/Navier-Stokes.

5.3.1 Une première approche pour sélectionner les solutions

Nous disposons d'un système d'équation linéaire $\mathbf{Ax} = \mathbf{b}$ permettant de calculer les poids de chaque voisins tout en s'assurant du respect des conditions de reproduction. Cependant, on remarque que si le nombre de particules dans le voisinage est supérieur à trois pour la première dérivée ou six pour la seconde dérivée, le système est dit sous-déterminé. En effet, on dispose d'une infinité de solutions. Il y a trop de poids à attribuer et trop peu de conditions. Il nous faut privilégier certaines solutions. Nous choisissons parmi cette infinité selon un critère ; un premier critère est la norme au carré. Pour cela, Colin, Egli et Serghini [CESM11] minimisent la norme au carré du vecteur de poids $\mathbf{x}$ afin d'obtenir une solution unique.

$$\min \frac{\|\mathbf{x}\|^2}{2}$$
$$\text{sujet à } \mathbf{Ax} = \mathbf{b}$$

La méthode des multiplicateurs de Lagrange permet justement de trouver les points stationnaires (ici les minimums) d'une fonction dérivable avec une telle contrainte. Pour cela il nous faut intégrer les multiplicateurs au sein du système linéaire à résoudre :

$$\begin{bmatrix} \mathbf{I}_n & \mathbf{M}^T \\ \mathbf{M} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{W} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b} \end{bmatrix}, \tag{5.15}$$

où $\mathbf{W}$ est le vecteur des solutions, contenant les poids souhaités. λ représente les coefficients des multiplicateurs de Lagrange.
On se retrouve donc à résoudre l'ensemble des équations suivantes :

$$\begin{cases} \mathbf{I}_n\mathbf{W} + \mathbf{M}^T\lambda = \mathbf{0}, \\ \mathbf{M}\mathbf{W} = \mathbf{b}. \end{cases} \tag{5.16}$$

L'usage des multiplicateurs de Lagrange nous permet donc d'obtenir une solution. La résolution du système est opérée à l'aide de la fonction `dgesv` fournie par Lapack [ADO92], qui réalise une décomposition LU.

Vérification de la résolution à l'aide de Lapack

À l'image des résultats numériques observés dans l'article [BBCE09], nous avons réalisé une étude sur la résolution de Lapack. Nous construisons une séquence de N points avec $N = 2000$ disposés aléatoirement (avec une distribution de probabilité uniforme) dans un domaine $\mathbb{R}^2$ avec $\mathbb{R} \in [0, 1]$. Nous avons fixé le voisinage de chaque particule de référence. Le but est d'étudier l'erreur de résolution du système fournie par Lapack. L'erreur est obtenue avec $\mathbf{Ax} - \mathbf{b}$, la moyenne des erreurs et l'erreur maximum obtenue sur les 2000 particules sont présentes dans le tableau 5.2.

	Première dérivée en x et y		Deuxième dérivée en x et y	
Nombre de particules	**Erreur moyenne**	**Erreur maximum**	**Erreur moyenne**	**Erreur maximum**
6	2,25E-17	1,78E-15	4,79E-14	2,30E-10
7	2,05E-17	1,78E-15	4,09E-16	7,11E-13
8	1,92E-17	5,55E-16	1,29E-16	6,04E-14
9	1,81E-17	4,51E-16	9,58E-17	8,53E-14
10	1,45E-17	6,11E-16	7,47E-17	1,20E-14
15	1,05E-17	6,66E-16	4,52E-17	3,11E-15
20	9,18E-18	7,77E-16	3,62E-17	3,11E-15
50	8,29E-18	1,11E-15	2,79E-17	4,00E-15
100	5,86E-18	1,15E-15	1,36E-17	4,66E-15

Tableau 5.2 – Tests de vérification des solutions

Il est important de noter que dans le cadre des simulations nous utilisons le type `double`, type à virgule flottante à précision double. Il s'agit habituellement du type IEEE-754 64 bits. On remarque que même si le niveau d'erreurs reste négligeable, il est préférable d'employer des simulations avec un nombre de voisins supérieur à huit.

Pour cela nous avons défini la valeur du support de noyau (théorique) égale à trois fois la valeur du rayon de la particule. Nous souhaitons obtenir un voisinage de neuf particules comme on peut le constater sur la figure 5.3.

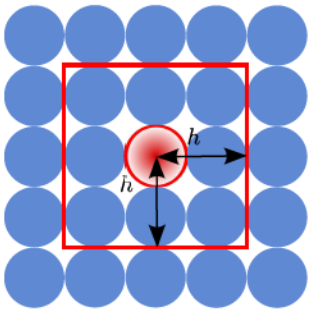


Figure 5.3 – Définition du support théorique

Ce même voisinage fournit d'excellents résultats en méthode SPH. Nous essayons d'obtenir des résultats comparables. Le h ne doit pas être trop grand pour des simulations similaires car nous perdrions l'avantage de l'usage des noyaux constants par morceaux. Cependant, dans l'optique de toujours résoudre le système, nous augmentons temporairement le support d'un noyau pour une particule donnée.

5.3.2 Cas et procédures de validation

Dans le but de comprendre le fonctionnement des noyaux CSTPM nous avons développé un cas pédagogique. Ce, cas visible sur la figure 5.4, sera très utile pour l'étude du calcul des poids. Ce domaine dispose seulement de 25 particules. Nous pouvons donc nous assurer, dans le cas où l'on souhaite favoriser certaines particules (sous-section 5.4.2), de la bonne disposition et répartition des poids sur les particules voisines.

Figure 5.4 – Présentation du cas pédagogique

Nous souhaitons fournir une simulation simple sans calculs de frontière. Nous souhaitons également que la densité initiale au sein du domaine soit uniforme. Pour cela la configuration géométrique du tore est utile («torus» en anglais). La figure 5.5 ci-après décrit comment nous avons procédé pour réaliser un tore avec un domaine en deux dimensions. Il est important de rappeler que dans ce cadre de simulations la gravité n'est pas présente :$\vec{g} = [0, 0]^T$.

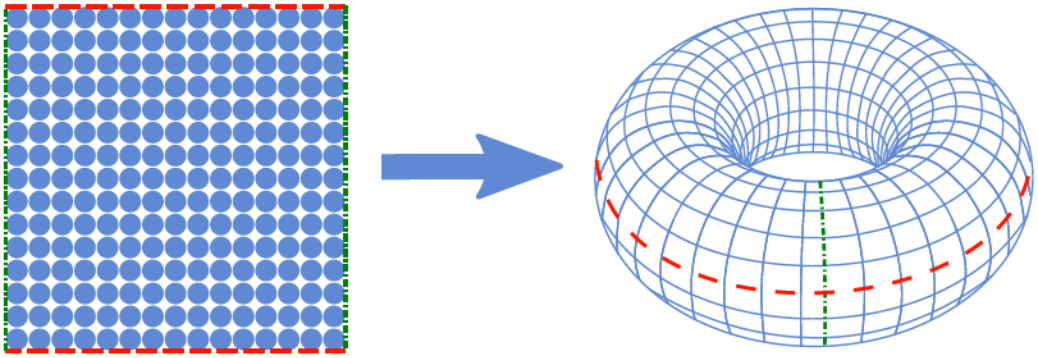


Figure 5.5 – Illustration de la conception du tore géométrique

5.3.3 Estimation de la densité

La densité définie pour une particule i, notée ρ_i est calculée de la manière suivante :

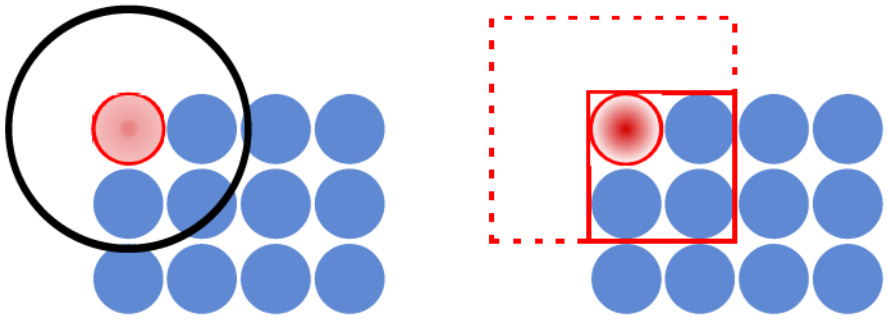
$$\rho_i = \sum_{j=1}^{n} \rho_j W_{cstpm}, \tag{5.17}$$

avec W_{cstpm} les poids obtenus pour estimer la fonction ρ avec la matrice $\mathbf{M}^{(0,1)}$ (5.5) et les conditions de reproduction (5.8). On retrouve une formulation similaire à l'estima-

tion de densité de manière SPH (3.4). Cette formulation (5.17) nécessite d'initialiser la densité au préalable. Pour cela nous initialisons la densité selon :

$$\rho_{init} = \frac{K}{4h^2}, \tag{5.18}$$

où K représente le nombre maximal de particules au sein du noyau qui est de neuf avec un support noyau donné par $h = 0.045$. On obtient donc une densité initiale $\rho_{init} = 1111$. Le principe de l'estimation de densité est comme pour la méthode SPH, d'observer les zones de forte densité et les zones de faible densité.



(a) Estimation de la fonction de densité à l'aide d'un noyau SPH

(b) Estimation de la fonction de densité à l'aide d'un noyau CSTPM

Figure 5.6 – Illustration de l'estimation de la fonction de densité

Toutefois nous n'obtenons pas les mêmes résultats comparativement à la méthode SPH. En effet, la faculté au décentrage des noyaux constants par morceaux oblige à ne disposer que des informations définies sur les particules (figure 5.6b) alors que la méthode SPH comme on peut le constater dispose de son support circulaire fixe (figure 5.6a). Ce support circulaire permet de connaître les zones de faible densité, par exemple à la frontière d'un liquide. Alors que l'usage de l'équation (5.17), nous fournit, pour l'ensemble des particules, la même valeur de densité. L'estimation de la densité obtenue grâce au noyau constant par morceaux est viable mathématiquement mais elle se fait de manière trop locale et on ne dispose que de peu d'informations spatiales autour des particules.

Pour remédier à cette difficulté nous avons décidé d'estimer la densité en s'appuyant sur la conservation de la masse. L'équation de la conservation de la masse

(1.26) fournit l'évolution de la densité à l'aide de la divergence du champ de vecteurs vitesse. La formulation SPH qui estime la densité en appliquant l'équation de la conservation de la masse est introduite dans la sous-section 3.1.1, il s'agit de l'équation (3.9). La formulation de la conservation de la masse est légèrement différente :

$$\begin{aligned}\frac{d\rho}{dt} &= -\nabla \cdot (\rho \vec{v}), \\ &= -\nabla\rho \cdot \vec{v} - \rho \nabla \cdot \vec{v}.\end{aligned} \tag{5.19}$$

En appliquant l'approximation des noyaux constants par morceaux, pour une particule i, on a :

$$\begin{aligned}\frac{d\rho_i}{dt} &= \sum_{j=1}^{n} -\rho_j \nabla W_{cstpm} \cdot \vec{v}_i - \rho_i \sum_{j=1}^{n} \vec{v}_j \cdot \nabla W_{cstpm} \\ &= \sum_{j=1}^{n} - \begin{bmatrix} \rho_j \partial W_{cstpm}/\partial x \\ \rho_j \partial W_{cstpm}/\partial y \end{bmatrix} \cdot \begin{bmatrix} \vec{v}_{x_i} \\ \vec{v}_{y_i} \end{bmatrix} - \rho_i \sum_{j=1}^{n} \begin{bmatrix} \vec{v}_{x_j} \\ \vec{v}_{y_j} \end{bmatrix} \cdot \begin{bmatrix} \partial W_{cstpm}/\partial x \\ \partial W_{cstpm}/\partial y \end{bmatrix}.\end{aligned} \tag{5.20}$$

Avec une intégration d'Euler (section B.2), on obtient la mise à jour de la densité pour le pas de temps suivant. Cependant, la densité évolue seulement dans le cas où il y a divergence des vélocités. Pour cela nous avons mis en place un souffleur constant («blower» en anglais) visible sur la figure 5.7 au centre de la représentation du tore.

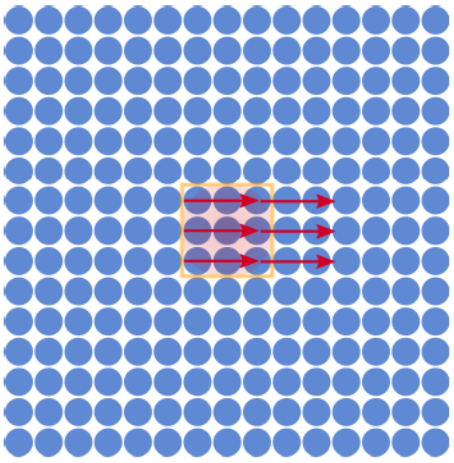


Figure 5.7 – Mise en place d'un souffleur

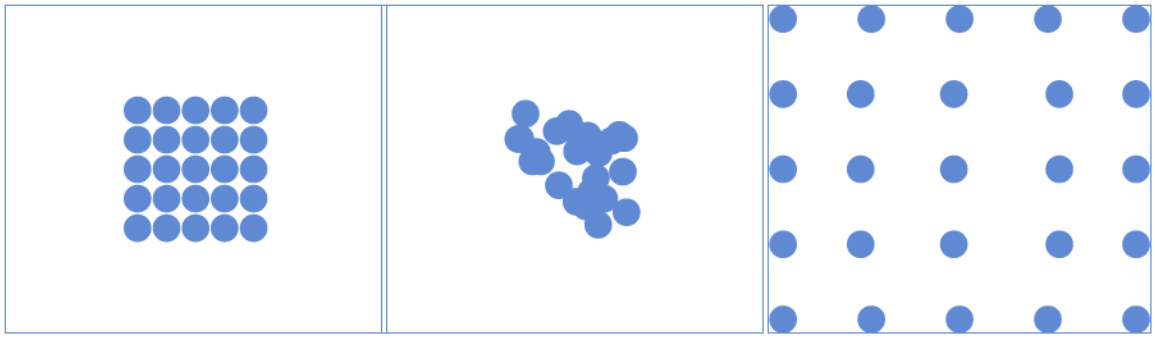
Les résultats observés sont très concluants et permettent de localiser les zones de faible et forte densité.

5.3.4 Estimation du gradient de la pression

La formulation de l'estimation du gradient de la pression au sein des équations de Euler/Navier-Stokes avec les noyaux CSTPM est la suivante :

$$\left\{-\frac{1}{\rho}\nabla p\right\}_i = -\frac{1}{\rho_i}\sum_{j=1}^{n} p_j \nabla W_{cstpm}. \tag{5.21}$$

Cette équation, ci-dessus, est très similaire à la formulation simple de l'estimation du gradient de la pression suivant la méthode SPH (3.15). L'obtention de la valeur de p est réalisée en utilisant une équation d'état (définie dans la sous-section 3.1.2), plus précisément l'équation des gaz parfaits à savoir l'équation (3.13). Les tests réalisés sur le cas d'école/pédagogique avec la méthode SPH mettent en lumière l'importance de la force de pression. Précédemment constaté au sein du chapitre 3, si la pression relative est initialement négative pour l'ensemble des particules on constate une force d'attraction (figure 5.8b). Si la pression relative est initialement positive pour l'ensemble des particules on observe une force de répulsion (figure 5.8c).



(a) Configuration de base des particules au sein du domaine

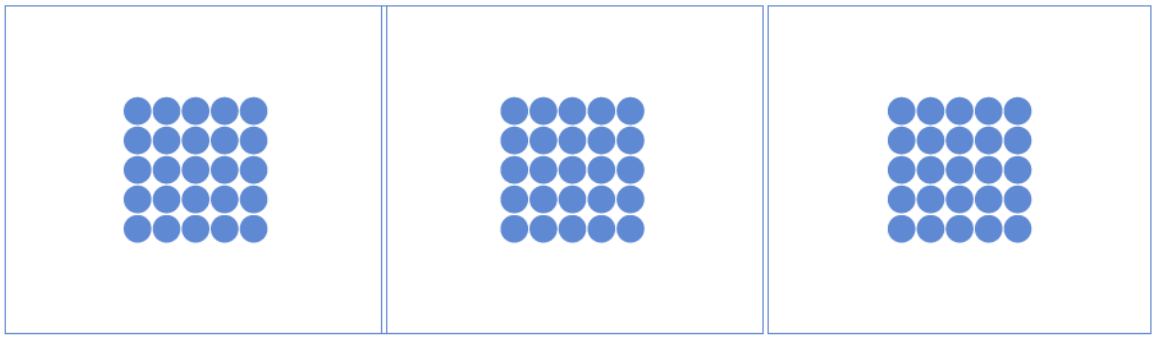
(b) Initialisation négative de la valeur de la pression relative

(c) Initialisation positive de la valeur de la pression relative

Figure 5.8 – Comportement de la force de pression de manière SPH

Les tests réalisés sur le cas pédagogique avec la méthode des noyaux CSTPM

démontrent également l'aspect trop local de la résolution du gradient de la pression. Si la pression relative est initialement négative ou initialement positive, on observe aucun changement sur la configuration des particules figure 5.9b et figure 5.9c. La force de pression intervient seulement dans le cas ou il y a une différence de pression présente au sein du support du noyau. Cela complique le développement de méthodes itératives appliquées aux noyaux CSTPM, car ces méthodes utilisées avec les noyaux SPH itèrent sur le principe d'attraction et de répulsion de cette force de pression. L'aspect restreint de la résolution du gradient de la pression peut conduire



(a) Configuration de base des particules au sein du domaine

(b) Initialisation négative de la valeur de la pression relative

(c) Initialisation positive de la valeur de la pression relative

Figure 5.9 – Comportement de la force de pression à l'aide d'un noyau CSTPM

à des regroupements de particules («clustering» en anglais). Dans une simulation lagrangienne, on ne souhaite pas de regroupement de particules. C'est pour cela que Müller [MCG03], par exemple, utilise des noyaux ne respectant pas correctement les conditions de reproduction des polynômes et d'interpolations : «Spiky».

5.3.5 Estimation de la viscosité

La formulation de l'estimation de la viscosité au sein des équations de Euler/Navier-Stokes avec les noyaux CSTPM est la suivante :

$$\left\{\frac{\mu}{\rho}\nabla^2\vec{v}\right\}_i = \frac{\mu}{\rho_i}\sum_{j=1}^{n} v_j \nabla^2 W_{cstpm}, \tag{5.22}$$

$$= \frac{\mu}{\rho i}\sum_{j=1}^{n}\begin{bmatrix}\vec{v}_{x_j}\partial^2 W_{cstpm}/\partial x^2 + \vec{v}_{x_j}\partial^2 W_{cstpm}/\partial y^2 \\ \vec{v}_{y_j}\partial^2 W_{cstpm}/\partial x^2 + \vec{v}_{y_j}\partial^2 W_{cstpm}/\partial y^2\end{bmatrix}. \tag{5.23}$$

Cependant, la contrainte de posséder un voisinage de six particules non alignées afin de résoudre le système avec $\mathbf{M}^{(2)}$ (5.6) peut être problématique. De plus nous l'avons vu dans les sections précédentes, l'usage des dérivées secondes (Laplacien) du noyau peut conduire à des erreurs d'approximations. Pour cela, Monaghan et d'autres auteurs de la méthode SPH ont recours à une viscosité souvent artificielle calculée avec le gradient du noyau et non son Lapalacien. Nous utilisons la correction de type XSPH (sous-section 3.1.5) à l'aide d'un noyau SPH pour stabiliser la simulation. L'usage de ce type de correction pour illustrer la viscosité peut permettre d'éviter le regroupement des particules. Nous obtenons de meilleurs résultats avec l'usage de la correction XSPH.

5.3.6 Amélioration de la simulation

Dans un souci de stabilisation de la simulation, nous avons introduit une force (artificielle) de collision qui influence fortement le comportement du fluide pour conserver une certaine cohésion particulaire (moléculaire) :

$$\vec{f}_C \tag{5.24}$$

Cette force de collision s'ajoute au calcul de la quantité du mouvement des equations de la dynamique des fluides (Navier-Stokes). Elle a pour but de stabiliser la simulation pour éviter les regroupements de particules pouvant conduire à des erreurs de résolutions du système ou encore des erreurs d'estimations. Cette force permet d'illustrer le fait que chaque particule dispose de son unité de volume. Précédemment, nous avons observé dans la sous-section 5.2.3 l'absence du terme $V_j = m_j/\rho_j$. À l'aide des

forces de collision nous pouvons définir une force permettant d'illustrer cette unité de volume implicite dans la valeur des poids. Chaque particule dispose de son volume fixe, surface en deux dimensions, provoquant des calculs de collisions. Deux particules ne peuvent occuper la même surface.

La force de collision interparticulaire est calculée en appliquant la méthode des éléments discrets (DEM) [Mis03]. On distingue la force d'élasticité $\vec{f}_{i,e}$, la force d'amortissement $\vec{f}_{i,a}$ et la force de cisaillement $\vec{f}_{i,x}$. Toutes ces forces représentent la force de collision notée $\vec{f}_C$:

$$\vec{f}_{i,e} = -\kappa(d - \|\vec{r}_{ij}\|)\frac{\vec{r}_{ij}}{\|\vec{r}_{ij}\|}, \tag{5.25}$$

$$\vec{f}_{i,a} = \tau\vec{v}_{ij}, \tag{5.26}$$

$$\vec{f}_{i,x} = \eta\vec{v}_{ij_T}, \tag{5.27}$$

où d, κ, τ et η représentent respectivement le diamètre de la particule de référence, le coefficient d'élasticité, d'amortissement et de cisaillement. La force de cisaillement est calculée avec la vitesse relative tangentielle. Avec $\vec{v}_{ij_T}$:

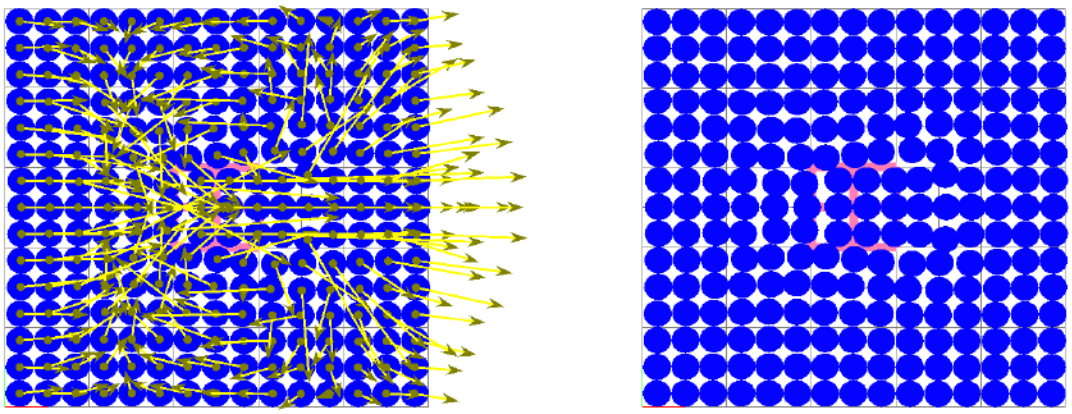
$$\vec{v}_{ij_T} = \vec{v}_{ij} - \left(\vec{v}_{ij} \cdot \frac{\vec{r}_{ij}}{\|\vec{r}_{ij}\|}\right)\frac{\vec{r}_{ij}}{\|\vec{r}_{ij}\|}. \tag{5.28}$$

Ensuite, la force de collision est appliquée pour chaque particule i :

$$\vec{f}_{i,C} = \sum_{j=1}^{n}\left(\vec{f}_{i,e} + \vec{f}_{i,a} + \vec{f}_{i,x}\right). \tag{5.29}$$

Les résultats observés à l'aide de cette force de collision sont très encourageants (figure 5.10 et figure 5.11). Cette méthode nous fournit un champ de vecteurs vitesses stable au sein du tore (figure 5.11a) et répondant à un comportement similaire à un fluide (figure 5.10a). Cette force est calculée au moment de la mise à jour du vecteur vitesse en fin de chaque pas de temps, pour ne pas trop influencer les calculs de pression et de viscosité. Nous avons défini d à 90% de sa valeur réelle ($d = 0.015$) permettant ainsi un léger regroupement visible sur la figure 5.10b et la figure 5.11b.

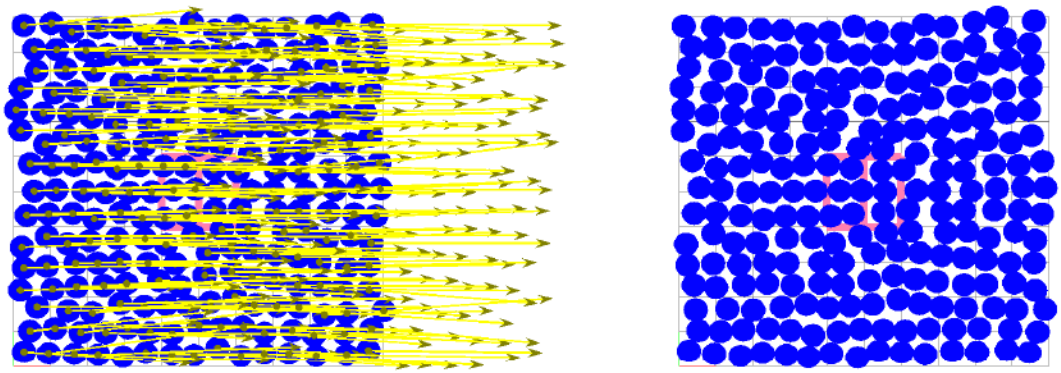
L'un des avantages majeurs de l'utilisation de cette force est qu'elle permet, non



(a) Champ de vecteurs vitesses (b) Position de chaque particule

Figure 5.10 – Illustration du début de la simulation de fluides au sein du tore

seulement une stabilisation de la simulation et des systèmes à résoudre, mais elle permet à notre approche de plus facilement respecter la condition de stabilité CFL (section B.1). En effet, une particule disposant d'une vitesse trop grande pour le pas de temps défini se voit rapidement diminuée par une collision avec une autre particule. Cependant l'usage d'une telle force non nulle (localement) influence fortement les equations de la dynamiques des fluides.



(a) Champ de vecteurs vitesses (b) Position de chaque particule

Figure 5.11 – Illustration de la fin de la simulation de fluides au sein du tore

5.4 Une deuxième approche pour sélectionner les solutions

5.4.1 Choisir par critère d'optimisation

Le titre paraphrase un article phare d'Emmanuel Candès [CT05]. En présence d'un système d'équations linéaires sous-déterminé, il faut bien choisir parmi la variété affine des solutions. Si on minimise la norme Euclidienne au carré, on aura la projection de l'origine sur la variété affine. Un autre choix est de minimiser la norme L_1 et non la norme L_2 précédemment utilisée. La situation est donc la suivante : nous avons un système d'équations linéaires $\mathbf{Ax} = \mathbf{b}$ permettant de calculer les poids de chaque voisins tout en s'assurant du respect des conditions de reproduction. Ce système est généralement sous-déterminé et nous devons sélectionner une solution particulière. Pour cela on définit une fonction objectif $\phi(x)$ et on souhaite résoudre :

$$\begin{aligned} &\min \phi(x), \\ &\text{sujet à } \mathbf{Ax} = \mathbf{b}. \end{aligned}$$

Voici quelques choix pour ϕ :

1. $\phi(x) = \|\mathbf{x}\|^2 = \sum_j |w_{i,j}|^2$. On parle alors de norme L_2 au carré, fonction strictement convexe et différentiable. Les conditions d'optimalité de Lagrange mènent à la définition d'un système d'équations linéaires (équation (5.15)) que l'on peut résoudre avec n'importe quelle routine d'algèbre linéaire, par exemple Lapack. La solution est toujours unique.
2. $\phi(x) = \sum_j \alpha_j |w_{i,j}|^2$; similaire à la précédente, mais pondérée chaque composante de $\mathbf{x}$ par α.
   - Si $\alpha > 0$, ça correspond à une norme dont les lignes de niveau sont des ellipses le long des axes canoniques, solution unique.
   - Si certains α_j sont nuls, les composantes correspondantes $w_{i,j}$ ne sont pas minimisées, donc on dit que leurs poids seront favorisés (pas défavorisés). C'est ici qu'intervient la notion de favorisation. On pourrait même essayer des valeurs négatives de α, et alors les composantes correspondantes $w_{i,j}$

seraient maximisées. ϕ n'est pas une norme ni une norme au carré dans ce cas. L'unicité de la solution n'est plus garantie si un ou plusieurs des α_j sont nuls.

3. $\phi(x) = \|\mathbf{x}\| = \sum_j |w_{i,j}|$. C'est la minimisation de la norme dite L_1. L'article de Candès [CT05] précité a rendu cette approche très populaire en y ajoutant du traitement de signal pour représenter des signaux de manière très éparse. Ce genre de choix réduit habituellement le nombre de poids positifs (non-nuls) et donnerait, probablement, une favorisation implicite. L'idée est de remplacer astucieusement $\min \sum_j |w_{i,j}|$ par $\min \sum_j u_{i,j}$ où $u_{i,j}$ modélise les valeurs absolues par les contraintes $u_{i,j} \geq w_{i,j}$ et $u_{i,j} \geq -w_{i,j}$

Nous décidons donc d'opter pour une programmation dite linéaire. Afin de résoudre :

$$\begin{aligned} &\min \|\mathbf{W}\| \\ &\text{sujet à } \mathbf{MW} = \mathbf{b} \end{aligned}$$

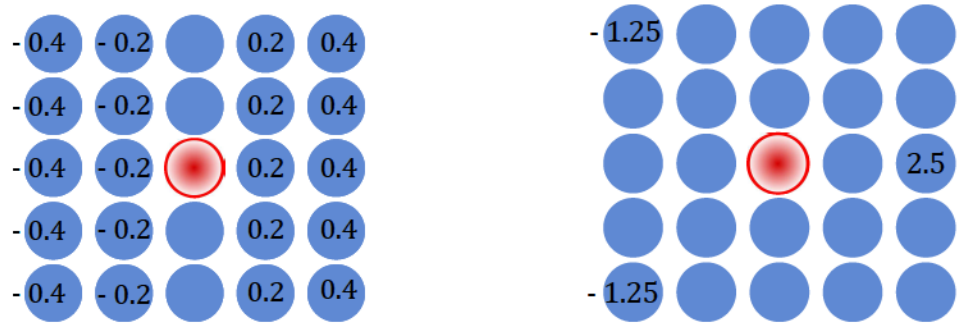
Le système devient donc :

$$\min_{\mathbf{U},\mathbf{W}} \sum u_{i,j} \tag{5.30}$$

$$\text{sujet à } \begin{bmatrix} \mathbf{0} & \mathbf{M} \\ -\mathbf{I}_n & -\mathbf{I}_n \\ -\mathbf{I}_n & \mathbf{I}_n \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{W} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{5.31}$$

La boite à outils «quapro» («linear quadratic programming solver» en anglais) [CSB09] de Scilab comporte une fonction `linpro` («linear programming solver» en anglais) qui résout directement ce genre de problème en autant qu'il soit ramené à la forme adéquate. Cependant même si cette boite à outils nous fournit des résultats concluants (figure 5.12b), il devient difficile d'intégrer Scilab au sein du programme C++. Pour cela, nous décidons d'utiliser «GLPK» [Mak00] («GNU linear programming kit» en anglais), qui est un logiciel destiné à résoudre des problèmes de programmation linéaire (LP) à grande échelle et d'autres problèmes connexes. Il s'agit d'un ensemble de routines écrites en C et organisées sous la forme d'une bibliothèque.

À l'aide de GLPK, nous obtenons les même résultats que pour la figure 5.12b via un algorithme du simplexe. On constate, sur la figure 5.12, une favorisation implicite en utilisant la programmation linéaire. Cette approche favorise un triplet de particules.



(a) Poids obtenus pour $\partial W_{cstpm}/\partial x$ avec le critère de norme L_2 au carré

(b) Poids obtenus pour $\partial W_{cstpm}/\partial x$ avec le critère de norme L_1

Figure 5.12 – Comparaison entre différentes méthodes de résolution pour $\partial W_{cstpm}/\partial x$

Comparaison des performances de calcul

Dans un contexte d'obtention de simulations dites en temps-réel, nous avons comparé les performances de calcul entre GLPK et Lapack. Au cours d'une simulation nous avons un certain nombre d'échantillons appelés particules et pour chaque particule à chaque pas de temps nous devons résoudre des systèmes linéaires. Il s'agit d'un total de cinq systèmes linéaires à résoudre et ce dans le but d'obtenir des poids pour chaque particule permettant d'approximer la fonction et ses deux premières dérivées. Le tableau 5.3, ci-dessous, permet de différencier le coût calculatoire de l'utilisation d'une bibliothèque intégralement écrite en C (GLPK) et d'une bibliothèque écrite en FORTRAN (Lapack) interfacé en C.

Une simple résolution d'un système avec GLPK coûte entre six et sept fois plus de temps. Ce coût calculatoire est normal, mais il nous pousse à réaliser des simulations dites hors ligne («off-line» en anglais).

Nombre de particules	Temps de calcul avec Lapack (ms)	Temps de calcul avec GLPK (ms)
500	60 ms	391 ms
1000	100 ms	812 ms
5000	493 ms	3514 ms
10000	1119 ms	7005 ms
15000	1666 ms	10486 ms
20000	2264 ms	14064 ms
25000	2648 ms	17572 ms

Tableau 5.3 – Tests réalisés pour comparer les performances de résolution de Lapack et GLPK

5.4.2 Solutions favorisées

Nous avons vu précédemment que grâce à l'approche de programmation linéaire nous disposons d'une favorisation implicite, cependant nous souhaitons favoriser certaines particules. En favorisant des particules alignées selon une même direction proche de la particule considérée, nous sommes capables de déterminer des poids favorisant les estimations dans cette direction. L'intérêt de favoriser une direction est de sélectionner certaines informations parmi toutes celles présentes dans le voisinage. L'angle avec la direction favorisée et la distance entre la particule de référence et les particule voisines sont deux critères de sélection importants. Plus on s'éloigne de la particule cible, plus h augmente et la qualité des approximations baisse en conséquence. Le principe de favorisations dans l'usage des nouveaux noyaux a déjà été employé par [CESM11, Sam14].

Sur la figure 5.13, on retrouve le voisinage de neuf particules, l'idée est de concentrer les poids pour la première dérivée par rapport à x sur la particule à droite et à gauche de la particule de référence. Pour la première dérivée par rapport à y, nous concentrons les poids sur la particule en haut et en bas. Nous souhaitons appliquer ce principe de favorisation, dans le cadre d'une approche de programmation linéaire.

Variantes de la programmation linéaire : favorisation indirecte

Après quelques essais, il est apparu clairement que la modélisation utilisant la norme L_1 ne possède pas de solution unique : la norme L_1 n'est pas une fonction strictement convexe. En général, on s'attend à ce que la solution soit unique, mais pour des configurations relativement régulières, il y aura plusieurs solutions optimales.

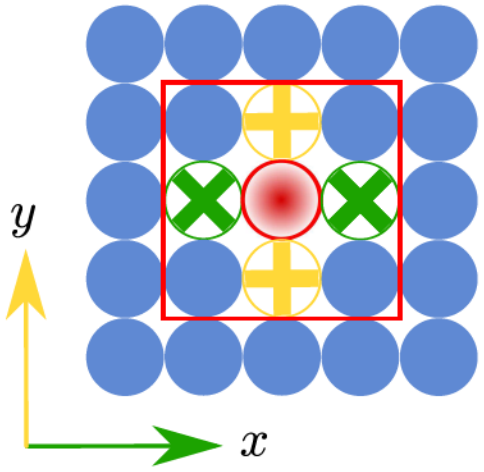


Figure 5.13 – Favorisation au sein du support

Nous souhaitons résoudre un système tout en favorisant certaines particules. Soit le système de programmation linéaire suivant :

$$\min \|\mathbf{W}\|,$$
$$\text{sujet à } \mathbf{MW} = \mathbf{b}.$$

Si ce problème est dégénéré et qu'il possède plusieurs solutions optimales, on peut toujours lui ajouter un critère additionnel dont le rôle sera d'encourager une des multiples solutions. Au lieu de résoudre $\min \|\mathbf{W}\|$, nous utilisons une norme pondérée $\min \|\mathbf{PW}\| = \min \sum|p_{i,j}w_{i,j}|$. Si on définit les $p_{i,j}$ à un, on retrouve le cas décrit dans l'équation (5.31). Nous avons fixé :

$$p_{i,j} = 1 + \beta_{i,j}, \tag{5.32}$$

où $\beta_{i,j}$ représente la distance euclidienne normalisée, séparant la particule de référence et la particule voisine. Une possibilité est d'adopter l'artifice de minimiser la somme

des variables auxiliaires $u_{i,j}$ dans le modèle, comme décrit précédemment :

$$\begin{aligned} \min \sum u_{i,j}, \\ \text{sujet à } u_{i,j} \geq p_{i,j} w_{i,j}, \\ u_{i,j} \geq -p_{i,j} w_{i,j}, \\ \mathbf{MW} = \mathbf{b}. \end{aligned}$$

Le système devient :

$$\begin{bmatrix} \mathbf{0} & \mathbf{M} \\ -\mathbf{I}_n & -\mathbf{P} \\ -\mathbf{I}_n & \mathbf{P} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{W} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}, \tag{5.33}$$

avec $\mathbf{P}$ la matrice diagonale des valeurs de $p_{i,j}$.

Il est facile de concevoir la distance euclidienne à partir de la matrice $M^{(0,1)}$ car elle dispose déjà de toutes les informations de différences de positions nécessaires. À l'aide de cette favorisation, nous n'obtenons pas vraiment le minimum de la norme L_1. Toutefois, nous obtenons des résultats encourageants qui respectent un comportement de fluide, visible sur la figure 5.13. Le système est nettement plus stable comparativement au critère de norme L_2 au carré. Cependant, l'usage des forces de collision est nécessaire si l'on souhaite une simulation inconditionnellement stable dans le temps.

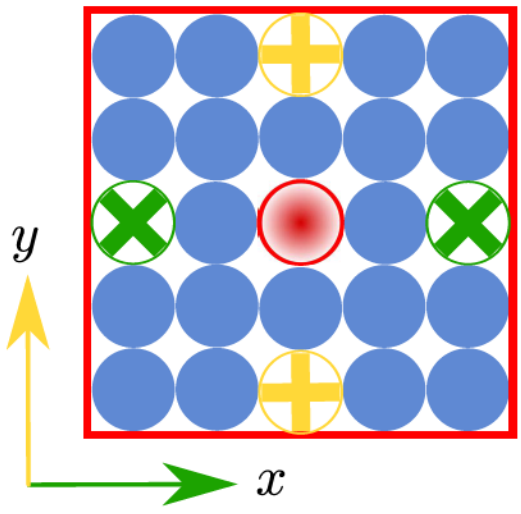


Figure 5.14 – Favorisation «indirecte» au sein du support dans le cas pédagogique

Cette première favorisation dispose d'un problème, visible sur la figure 5.14, il s'agit d'une favorisation «indirecte». Dans le cas où l'on souhaite agrandir le voisinage par exemple dans le cas pédagogique avec un voisinage de 25 particules, on constate que les particules favorisées sont les plus alignées, mais également les plus éloignées ce qui ne correspond pas à notre objectif de favorisation. On souhaite favoriser les particules les plus alignées et les plus proches de la particule concernée.

Variantes de la programmation linéaire : favorisation directe

Une autre possibilité est de continuer d'adopter l'artifice de minimiser la somme des variables auxiliaires $u_{i,j}$ dans le modèle, comme décrit précédemment :

$$\begin{aligned} \min \sum u_{i,j}, \\ \text{sujet à } u_{i,j} \geq p_{i,j} w_{i,j}, \\ u_{i,j} \geq -p_{i,j} w_{i,j}, \\ \mathbf{MW} = \mathbf{b}. \end{aligned}$$

Le système devient :

$$\begin{bmatrix} \mathbf{0} & \mathbf{M} \\ -\mathbf{I}_n & -\mathbf{P} \\ -\mathbf{I}_n & \mathbf{P} \end{bmatrix} \begin{bmatrix} \mathbf{U} \\ \mathbf{W} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{5.34}$$

En modifiant notre approche de l'usage des $p_{i,j}$, cette fois-ci, on définit les $p_{i,j}$ de la façon suivante :

$$p_{i,j} = \beta_{i,j}, \tag{5.35}$$

où $\beta_{i,j}$ représente la distance euclidienne normalisée.

Cette deuxième approche dispose d'une favorisation «directe». Contrairement à la figure 5.14, la figure 5.15 permet de visualiser les améliorations de notre deuxième approche. Les particules les plus alignées et les plus proches se retrouvent être favorisées. Nous obtenons une simulation stable après de nombreuses itérations (au sein du tore avec un souffleur) ce qui permet de s'affranchir de l'usage de la force de collisions. La simulation est inconditionnellement stable dans le temps sans l'usage de forces de collisions, visible sur la figure 5.16.

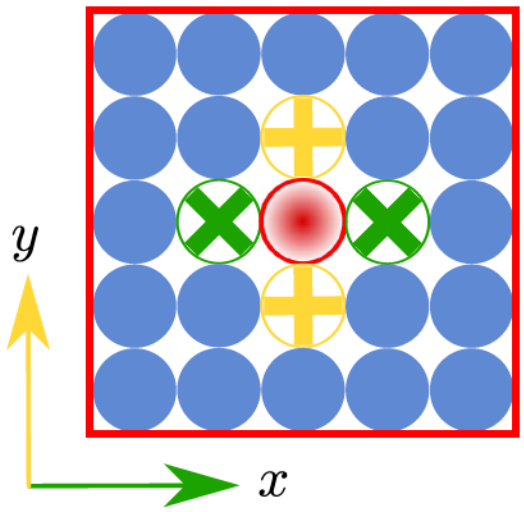
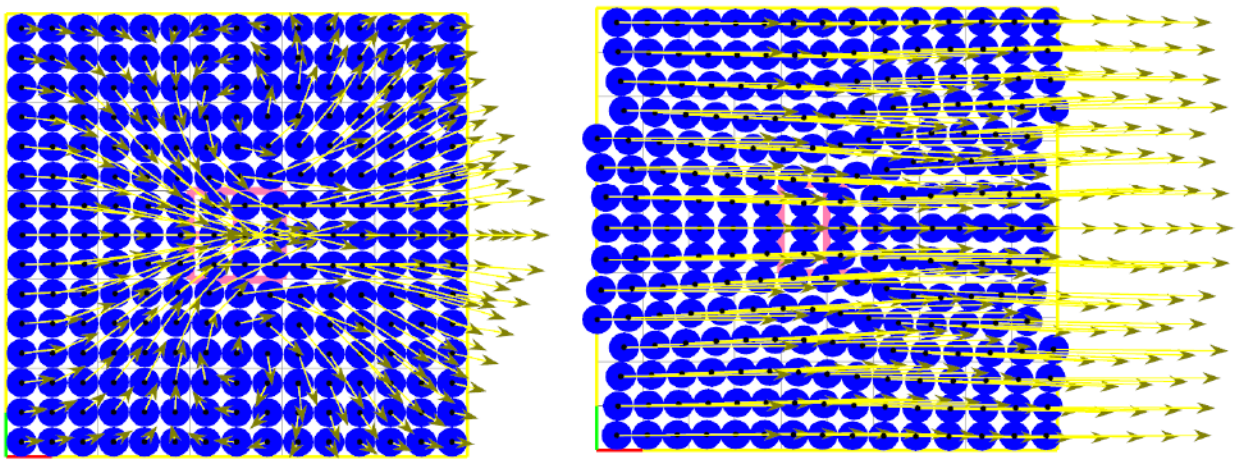


Figure 5.15 – Favorisation «directe» au sein du support dans le cas pédagogique



(a) Illustration du début de la simulation de fluides au sein du tore

(b) Simulation de fluides au sein du tore avec favorisation «directe»

Figure 5.16 – Illustration de la fin de la simulation de fluides au sein du tore

5.5 Conclusion

Au sein de ce chapitre, nous avons décrit et mis en œuvre des techniques permettant d'appliquer les noyaux constants par morceaux au sein d'un simulation totalement lagrangienne. L'approche de programmation linéaire fournit des résultats très encourageants. Une favorisation implicite et la possibilité de favoriser d'autres

particules fournissent de très bon résultats même si, dans un souci de stabilisation, nous sommes parfois contraint d'ajouter des forces de collision au sein du système. Nous avons démontré que l'usage d'une favorisation «directe» permet de s'affranchir des calculs des forces de collisions. Les avantages de la programmation linéaire serait susceptibles de fournir d'excellents résultats au sein de méthodes eulériennes. Cependant, la nécessité de résoudre de nombreux systèmes linéaires pour chaque particule influence fortement sur les performances globales de rendu de simulation.

Chapitre 6

Programmation parallèle sur GPU

Les informations contenues dans cette section sont en partie extraites du cours «Heterogeneous Parallel Programming» [Hwu14], du livre «Programming Massively Parallel Processors : A Hands-on Approach» [KWm12] et du guide de programmation de CUDA («Compute Unified Device Architecture» en anglais) [Nvi14].

6.1 Introduction à CUDA

6.1.1 Historique

Pendant trente ans, l'un des principaux moyens d'améliorer les performances des calculs sur ordinateur consistait à accroître la fréquence de l'horloge des processeurs, les unités centrales de traitement (CPU, «Central Processing Unit» en anglais). Durant les années 1990, la demande en graphisme en 3D s'est rapidement accrue. Des sociétés comme Nvidia, ATI Technologies ont commencé à produire des accélérateurs graphiques pour le grand public. Pour la première fois, un processeur graphique pouvait exécuter directement les traitements concernant les transformations géométriques et la lumière, permettant ainsi de créer des applications visuellement plus intéressantes. À cette époque est née l'unité de traitement graphique (GPU, «Graphics Processing Unit» en anglais). Avec cette apparition, de nombreux chercheurs ont étudié la possibilité d'utiliser ces unités de traitement pour autre chose que de l'affi-

chage de type OpenGL ou DirectX. Les débuts des traitements autres que graphiques sur le GPU étaient complexes. Toute tentative d'effectuer un traitement quelconque sur un GPU était soumise aux contraintes de l'API («Application Programming Interface» en anglais) graphique utilisée. En novembre 2006, Nvidia dévoila le premier GPU DirectX 10 : la GeForce 8800 GTX, qui fut également le premier GPU reposant sur l'architecture CUDA («Compute Unified Device Architecture» en anglais). Cette architecture a, en effet, introduit plusieurs nouveaux composants conçus spécialement pour les traitements GPU généraux et gommé ainsi la plupart des restrictions induites par les précédents circuits. Cette technologie de programmation dite GPGPU («General-Purpose computation on Graphic Processing Units» en anglais) exploite la puissance de calcul des GPU pour le traitement des tâches massivement parallèles.

Pour intéresser le maximum de développeurs possibles, Nvidia a donc choisi le langage C, une norme incontournable de l'industrie informatique. Depuis 2006 CUDA rencontre un fort succès. De nombreuses bibliothèques et applications voient le jour : «cuFFT–Fast» utilisée pour les transformées de Fourrier, «cuSPARSE» utilisée pour les matrices creuses *etc.* Depuis son lancement au début de 2007, un grand nombre de sociétés (Oracle Corporation, MathWorks *etc.*) choisissent de développer une partie de leurs applications en CUDA C. Les gains en terme de performance sont souvent de plusieurs ordres de grandeur (exemple dans la sous-section suivante) par rapport aux implémentations précédentes.

6.1.2 Pourquoi CUDA ? Pourquoi le GPU ?

La plateforme CUDA inclut des extensions C et C++ qui permettent l'expression de données denses et complexes dans un contexte de parallélisme. Les programmeurs peuvent choisir d'exprimer le parallélisme avec des langages à hautes performances comme C, C++, Fortran ou avec des standards ouverts comme les directives OpenACC («Open Accelerators» en anglais). L'usage de calcul parallèle est aujourd'hui déployée dans des milliers d'applications accélérées par les GPU et elle est présente dans de nombreux dossiers de recherche.

Cependant, il existe une autre API susceptible de répondre aux demandes de traitements parallèles : OpenCL («Open Computing Language» en anglais). OpenCL

est la combinaison d'une API et d'un langage de programmation dérivé du C, proposé comme un standard ouvert par Khronos Group depuis décembre 2008. Cette API plus jeune est multi-plateforme contrairement à CUDA qui ne fonctionne que sur les dispositifs Nvidia. Toutefois, disposant d'une carte Nvidia GeForce GTX 650 Ti de type Kepler, nous avons fait le choix de nous tourner vers l'API qui offre de meilleures performances et qui dispose d'un langage plus proche de l'architecture matérielle.

Les GPU sont d'ores et déjà utilisés dans de nombreux domaines. Leur utilité et efficacité n'est plus à démontrer. Par exemple, le National Center for Atmospheric Research aux Etats-Unis a porté 1% de son code de recherche et prévision météorologique sous CUDA et obtenu un gain de performance de 20% sur la totalité de l'application. En France, BNP Paribas a implémenté une architecture GPU qui contient deux modules Tesla S1070, consommant $2kW$ afin de remplacer environ 500 coeurs CPU, consommant $25kW$. Ceci offre :

- une division par 190 de l'ensemble de la consommation électrique ;
- des temps de réponses divisés par 15 ;
- réductions des coûts.

Alors que les CPU sont conçus pour exécuter un seul thread (anglicisme, fil d'exécution) contenant des instructions séquentielles à une cadence élevée. Les GPU quant à eux sont conçus pour exécuter des instructions en parallèle dans de nombreux threads. Notons qu'un thread sur un GPU n'a pas tout à fait le même sens qu'un thread CPU. Le choix d'une technologie GPGPU représentée majoritairement par Nvidia semble répondre parfaitement aux besoins nécessaires dans le cadre d'une simulation de fluide en temps réel. Une remise en cause des algorithmes de simulations (chapitre 4) afin de se tourner vers des technologies parallèles (HPC, «High-Performance Computing» en anglais) est nécessaire et permettra de meilleures performances.

6.2 Quelques notions sur CUDA

CUDA est composé d'un «Framework», d'un ensemble d'outils et d'une extension du langage C. Grâce à CUDA, le développeur peut utiliser la puissance de calcul d'une carte graphique pour certaines opérations destinées à être traitées par le GPU

au lieu du CPU. Le CPU est d'ailleurs toujours nécessaire pour coordonner le travail avec le GPU. Le GPU est ainsi vu comme un coprocesseur massivement parallèle très bien adapté au traitement d'algorithmes parallélisables. Une opération destinée au GPU est appelée un «kernel» (noyau, par la suite on conservera le terme de kernel). L'exécution d'un programme CUDA s'effectue de la façon suivante :

1. le programme est exécuté par le CPU ;
2. un kernel est invoqué, son exécution se déplace sur le GPU ;
3. un grand nombre de threads sont générés et exécutés en parallèle sur le GPU.

6.2.1 Modèle des threads sur CUDA

L'API CUDA permet de reproduire, au niveau logiciel, les spécificités de l'architecture matérielle GPU et de gérer la communication entre CPU et GPU. Cela permet de voir, à travers la programmation, le GPU comme une grille de calcul à une ou deux dimensions, formée de blocs de calcul indépendants visibles sur la figure 6.1. Chacun de ces blocs est physiquement lié à un multiprocesseur de flux (appelé «SM» ou «SMX» par Nvidia) et est décomposé en une matrice de threads à une, deux ou trois dimensions. C'est au développeur d'organiser les blocs sur la grille et de déterminer la dimension et la taille des blocs selon les caractéristiques de son application. Par exemple, si l'on souhaite multiplier deux matrices ou réaliser une opération sur des volumes, on choisira respectivement des blocs à deux dimensions ou à trois dimensions. La hiérarchie des threads permet de comprendre la cartographie des processeurs sur un GPU (caractéristiques sur une architecture GPU de type Kepler) :

1. le kernel est invoqué par le CPU et exécuté par le GPU. Les dimensions de la grille doivent être spécifiées au lancement du kernel. La grille, de dimension (x, y) est telle que $1 \leq x, y \leq 65536$;
2. chaque SMX ordonnance et exécute quatre warps de manière concurrente, soit 32 x 4 x 16 = 2048 threads exécutés en parallèle ;
3. le SMX peut traiter deux instructions indépendantes par warp, soit 2 x 4 = 8 instructions traitées en parallèle par cycle d'horloge, pour chaque SMX, d'où 32 pour le GPU utilisé (4 SMX).

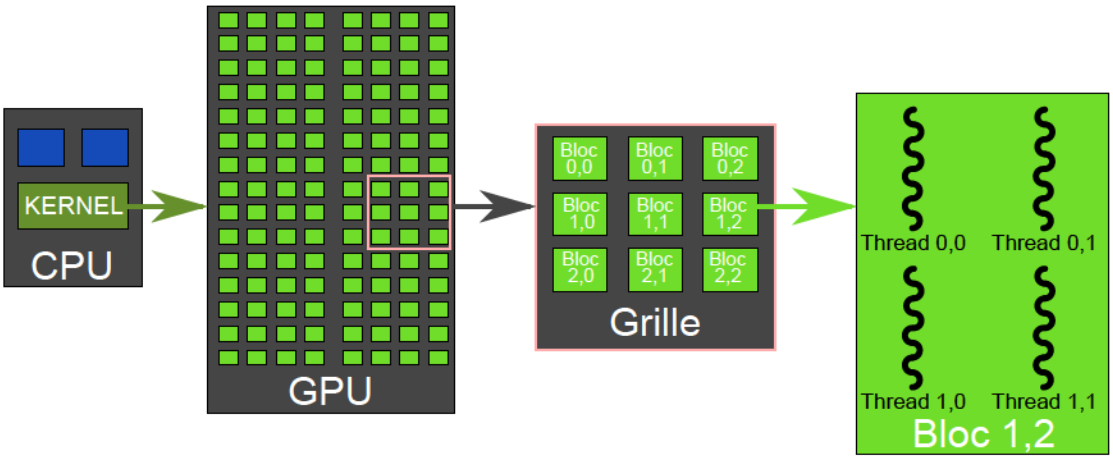


Figure 6.1 – Modèle simplifié des threads sur CUDA

6.2.2 Hiérarchie mémoire sur CUDA

Le principal goulot d'étranglement dans les traitements GPU est le plus souvent la bande passante du bus (système de communication partagé) de communication entre le CPU et le GPU [GH11]. Un GPU dispose d'une puissance de traitement telle qu'il est impossible de lui fournir les données suffisamment vite pour exploiter cette puissance. Il est donc nécessaire d'avoir des techniques permettant de réduire le trafic entre mémoire CPU et GPU.

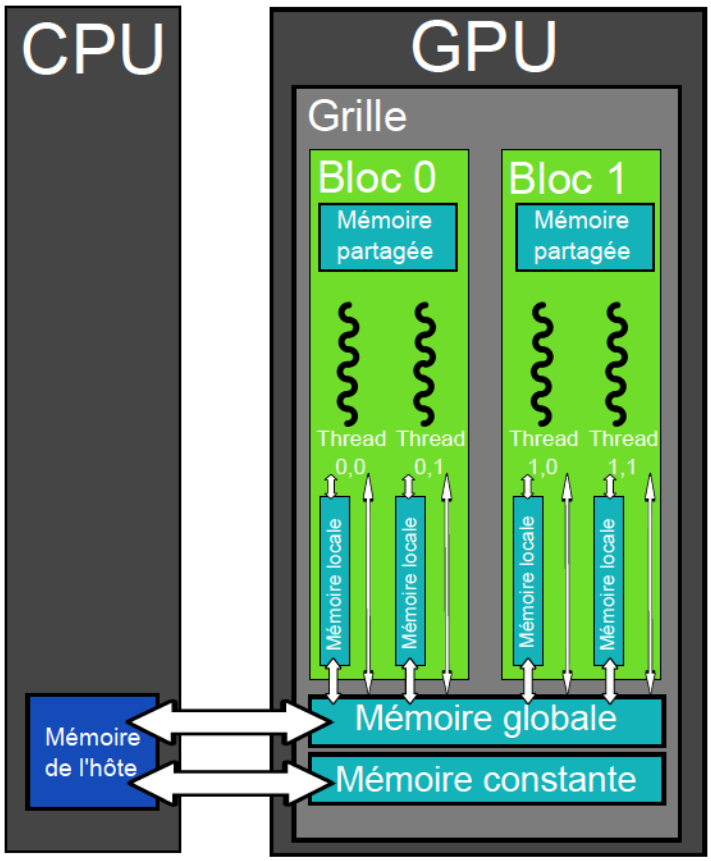


Figure 6.2 – Modèle simplifié de la mémoire sur CUDA

La description du modèle mémoire sur CUDA sur la figure 6.2 est simplifiée pour faciliter la compréhension.

- La mémoire globale en bas du schéma (figure 6.2) est le moyen de communiquer des données entre l'hôte et le GPU (souvent appelé «device» en anglais). Le contenu de la mémoire globale est visible depuis tous les threads et est accessible en lecture/écriture ;
- la mémoire constante indiquée «constant memory» en anglais, n'est accessible qu'en lecture seulement par le GPU ;
- la mémoire partagée par bloc est visible depuis tous les threads de ce bloc ;
- la mémoire locale, comme les registres, n'est visible que du thread.

6.2.3 Des temps d'accès mémoire différents

Comme on peut le constater sur la figure 6.2, la mémoire partagée est associée au bloc. Elle se situe sur la même puce que les cœurs («cores» en anglais), exécutant les threads. La communication est relativement rapide, car la mémoire vive statique (SRAM, «Static RAM» en anglais), est plus rapide que la mémoire située en dehors de la puce («off-chip» en anglais) de type mémoire vive dynamique (DRAM, «Dynamic RAM» en anglais). Il est donc important d'optimiser au maximum l'usage de la mémoire partagée au sein des algorithmes (usage décrit au sein de la sous-section 6.3.3). Chaque thread dispose d'un accès direct à ses registres et à sa mémoire locale. Les registres sont beaucoup plus rapides que la mémoire locale. L'accès d'un thread à la mémoire globale souffre des problèmes inhérents aux communications entre puces : consommation de puissance, débit, *etc.* La mémoire constante (accessible qu'en lecture) dispose d'un accès privilégié, elle devient très utile pour stocker les constantes de la simulation.

L'ensemble des informations décrites sont résumées au sein du tableau 6.1. Ce tableau montre la répartition des variables entre les différentes zones mémoires.

Déclaration	Lieu de stockage	Rapidité	Visibilité	Durée de vie
`int variable;`	un registre du thread	bonne	thread	thread
`int tableau[10];`	la mémoire locale du thread	moyenne	thread	thread
`__shared__ int variable;`	la mémoire partagée	bonne	bloc	bloc
`__device__ int variable;`	la mémoire globale	lente	grille	application
`__constant__ int constante;`	la mémoire constante	bonne	grille	application

Tableau 6.1 – Répartition des variables entre les différentes zones mémoires

6.3 Simulation lagrangienne sur le GPU

6.3.1 Un bon modèle de données

Une bonne structure de données

Le concept de particule porteuse d'informations et interagissant avec ses voisines, conduit naturellement à une représentation des données sous forme d'objets. Il faut adopter une structure de données adéquate en termes de performance. Deux choix s'offrent :

- une structure de tableaux, communément appelée SOA, «Structure Of Array» en anglais ;
- un tableau de structures, communément appelé AOS, «Array Of Structure» en anglais.

```
struct Particule
{
  float x, y, z, densite;
  float vx, vy, vz, pression;
};

Particule m_systeme[N];
```

algorithme 6.1 – Tableau de structures

```
struct SystemeDeParticules
{
 float x[N], y[N], z[N], densite[N];
 float vx[N], vy[N], vz[N], pression[N];
};

SystemeDeParticules m_systeme;
```

algorithme 6.2 – Structure de tableaux

Sur la figure 6.3, un simple algorithme de tri (tri par base sur le GPU) est appliqué sur x (générée aléatoirement), une des composantes de l'algorithme 6.1 et de l'algorithme 6.2. La taille de l'élément en abscisse représente N. En terme de performance, on gagne jusqu'à 5 fois plus de temps en adoptant une structure de type SOA comme le montre la figure 6.3.

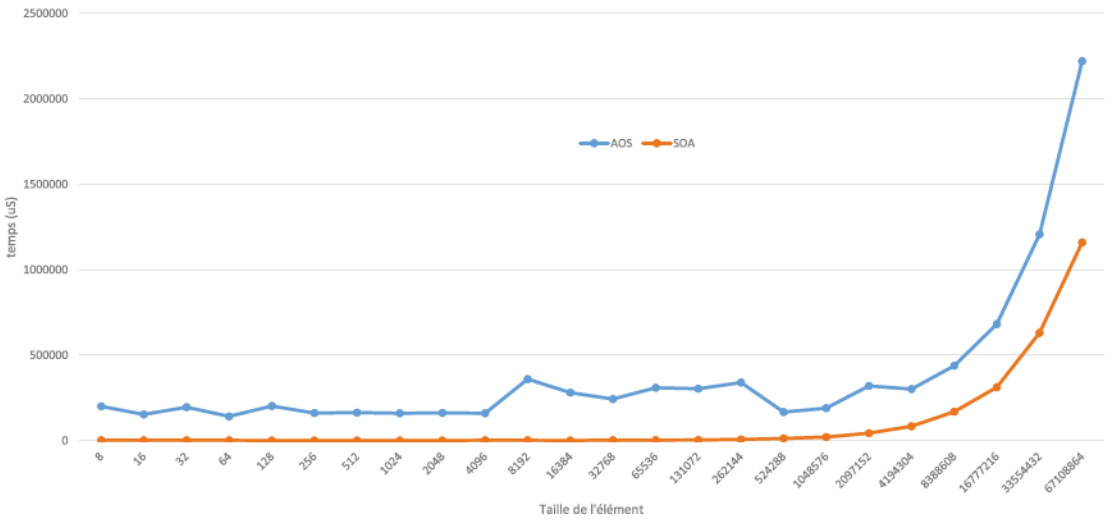


Figure 6.3 – Comparaison du temps d'exécution de l'algorithme de tri

Un bon type de données

```
struct SystemeDeParticules
{
 float4 x_y_z_densite[N];
 float4 vx_vy_vz_pressure[N];
};
```

algorithme 6.3 – Représentation du système de particules sur le GPU

L'usage au sein de l'algorithme 6.3 de `float4` est primordial (type défini par Nvidia [Nvi14] composé de 4 valeurs flottantes). Les échanges avec la mémoire globale sont de 128 octets alignés. Selon [NHP07], il est important d'utiliser des variables de type `float4` car ce type dispose de 16 octets alignés. Contrairement au type `float3` qui est représenté sur 12 octets non alignés, le type `float4` permet au SMX de mieux gérer les données en un seul cycle d'horloge. L'usage de `float3` aurait conduit à ce qu'on appelle des débordements mémoire et donc plusieurs transactions de 128 octets pour acquérir certaines données. De plus, ce type de données nous permet d'accéder aux coordonnées de la particule (x, y, z) en même temps que la valeur de sa densité. La meilleure approche est par conséquent de considérer ce qui engendre le moins de transactions de 128 octets.

6.3.2 Utilisation de fonctions GPU sur le CPU

Les fonctions GPU appelées kernel sont contenues au sein de fichiers sources contenant les extensions du langage CUDA (*.cu). Ces fichiers sources sont compilés avec nvcc (Nvidia CUDA compilateur). Pour plus de détails sur le compilateur CUDA, se référer à [Nvi14]. Les codes CUDA fonctionnent sur le CPU et le GPU. Cependant afin de l'intégrer au programme C++ nous utilisons les fonctions CUDA à l'aide des fonctions «wrapper». Une fonction «wrapper» est une fonction contenue dans un fichier dit «wrap» dont la fonction principale est d'appeler une autre fonction.

```
void calcul_index_lineaire_cpu(float4* x_y_z_densite, int* indexTri, int N)
{
  for (int id = 0; id<N; ++id)
  {
    int3 posPartGrille; //int3 (3 valeurs entieres)
    float3 positionParticule = float3(x_y_z_densite[id]);//recuperation des 3 premieres valeurs de float4
    posPartGrille.x = floor(positionParticule.x / tailleCaseX);//index cx de la position px de la particule
    posPartGrille.y = floor(positionParticule.y / tailleCaseY);//index cy de la position py de la particule
    posPartGrille.z = floor(positionParticule.z / tailleCaseZ);//index cz de la position pz de la particule
    // plus souvent tailleCaseX=tailleCaseY=tailleCaseZ=h
    indexTri[id] = posPartGrille.z * nbCasesY * nbCasesX + posPartGrille.y * nbCasesX + posPartGrille.x;
  }
}

void main()
{
.....
    calcul_index_lineaire(x_y_z_densite, indexTri, N);
}
```

algorithme 6.4 – Version CPU du code

```
__global__ void calcul_index_lineaire_gpu(float4* x_y_z_densite, int* indexTri, int N)
{
  int id = blockIdx.x * blockDim.x + threadIdx.x;
  if (id < N)
  {
    int3 posPartGrille;//int3 (3 valeurs entieres)
    float3 positionParticule = float3(x_y_z_densite[id]);//recuperation des 3 premieres valeurs de float4
    posPartGrille.x = floor(positionParticule.x / tailleCaseX);//index cx de la position px de la particule
    posPartGrille.y = floor(positionParticule.y / tailleCaseY);//index cy de la position py de la particule
    posPartGrille.z = floor(positionParticule.z / tailleCaseZ);//index cz de la position pz de la particule

    indexTri[id] = posPartGrille.z * nbCasesY * nbCasesX + posPartGrille.y * nbCasesX + posPartGrille.x;
  }
}
extern "C"
{
    void calcul_index_lineaire_for_cpp()
    {
    .....
        dim3 dimBlock (blocksize);//dim3 similaire a int3
        dim3 dimGrid ( ceil( N / (float)blocksize) );
        calcul_index_lineaire_gpu<<<dimGrid, dimBlock>>>(x_y_z_densite, indexTri, N);
    }
}
```

algorithme 6.5 – Version GPU du code utilisé au sein du programme C++

`blocksize` est le nombre maximal de threads qu'un bloc peut contenir (1024 threads pour notre GPU). Toutes les fonctions `__global__` et `__device__` ont automatiquement accès aux variables définies, suivantes :

- `dim3 gridDim`, dimensions de la grille, nombre de blocs (au plus 2D) ;
- `dim3 blockDim`, dimensions du bloc, nombre de threads ;
- `dim3 blockIdx`, index du bloc au sein de la grille ;
- `dim3 threadIdx`, index du thread au sein du bloc.

Ainsi l'algorithme 6.4 et l'algorithme 6.5 décrivent et comparent la programmation CPU et GPU d'une fonction de calcul d'index linéaire. Cette fonction sera très utile dans la recherche de voisinage (*cf.* section 6.3.3). On constate qu'il est simple d'utiliser une fonction GPU afin de remplacer une simple boucle `for`. En considérant une simulation de n particules, notée `N` dans le programme, on décide de définir une fonction kernel avec une grille unidimensionnelle. Chaque thread correspond à chaque particule. On souhaite obtenir une grille de taille suffisamment grande car chaque bloc ne peut contenir plus de 1024 threads.

Par exemple, pour une simulation avec `N=50000`, on impose `blocksize=1024` et donc avec la relation, `ceil( N / (float)blocksize)` on définit une grille de 49 blocs contenant chacun 1024 threads.

La mémoire constante est utilisée en lecture pour les constantes comme `tailleCaseX` ou encore `nbCasesX`. La lecture depuis cette mémoire ne coûte qu'un cycle. Pour tous les threads d'un demi-warp, la lecture depuis cette mémoire est aussi rapide que depuis un registre comme on peut le remarquer dans le tableau 6.1.

6.3.3 Une recherche de voisinage efficace

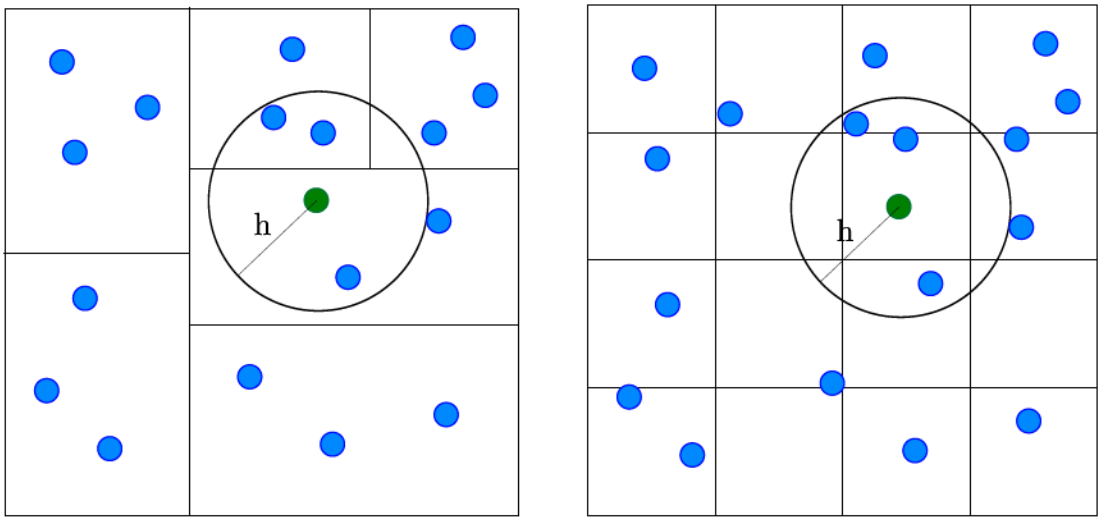
Structure en arbre ou grille uniforme

La recherche de voisinage, dans le cadre d'une simulation de fluide lagrangienne est un élément clé de la qualité et de l'aspect temps réel de la simulation (*cf.* chapitre 4). Si l'on a n particules, on ne souhaite pas un temps de calcul en $O(n^2)$. Afin de réduire ce temps de calcul, différentes méthodes de structure de données de partition de l'espace existent. Nous allons décrire deux grandes familles de structure de données de partition de l'espace, la structure en arbre («tree» en anglais) et la grille uniforme. Nous allons mettre l'accent sur deux caractéristiques de ces structures que sont la complexité du temps de construction et d'accès aux données. En effet, dans un contexte de simulation de particules, le voisinage évolue dynamiquement. Ceci nécessite une reconstruction de la structure à chaque pas de temps. On peut voir sur la figure 6.4 que la recherche de voisinage nécessite un accès rapide aux cellules adjacentes de la particule courante. De plus, nous devons conserver les positions des données présentes dans le domaine c'est-à-dire ne déplacer aucune particule. Les différentes complexité en temps de nos deux structures sont les suivantes :

	Complexité en temps en moyenne	
Type de structure	**Construction**	**Accès**
Structure en arbre	$O(n\,log\,n)$	$O(log\,n)$
Grille uniforme	$O(n)$	$O(1)$

Tableau 6.2 – Complexité entre une grille uniforme et une structure en arbre

D'après les publications [MCG03, CBP05, BT07, SB12, MM13, YT13], il est plus simple d'utiliser une structure de type grille car on souhaite privilégier l'accès aux données. Notre approche est d'utiliser une grille uniforme sur le GPU afin d'accélérer et de faciliter la recherche de voisinage.



(a) Une structure en arbre de type «Kd-tree» (b) Une grille uniforme

Figure 6.4 – Comparaison visuelle entre une grille uniforme et une structure en arbre

La grille uniforme simple

On remarque sur la figure 6.5, que la particule concernée disposant d'une fonction noyau de support h, nécessite un voisinage de 9 cases (de longueur de côté h) en comptant elle-même. Soit 27 cases si l'on se situe en trois dimensions. Chaque case doit avoir les côtés de taille égale au support du noyau, dans le but d'obtenir au moins toutes les particules situées à une distance h de la particule de référence.

On dispose donc d'une grille uniforme avec `nbCasesX`, `nbCasesY` et `nbCasesZ` qui représentent, respectivement, le nombre de cases en x, y et z. Nous devons déterminer l'index d'une particule au sein de la grille afin de récupérer les particules contenues dans les cases adjacentes. Cet index s'obtient facilement en divisant la position de la particule par la taille d'une case. Par la suite on utilise l'indexation linéaire qui nous permet de réduire en une seule dimension le problème à trois ou deux dimensions (fonction utilisée au sein de l'algorithme 6.5).

Par exemple, pour une particule de position (px, py, pz), on en déduit qu'elle se situe au sein de la case (cx, cy, cz). Son index linéaire est le suivant :

```
int IdxLineaire = cx + cy * nbCasesX + cz * nbCasesX * nbCasesY
```

algorithme 6.6 – Index linéaire d'une particule

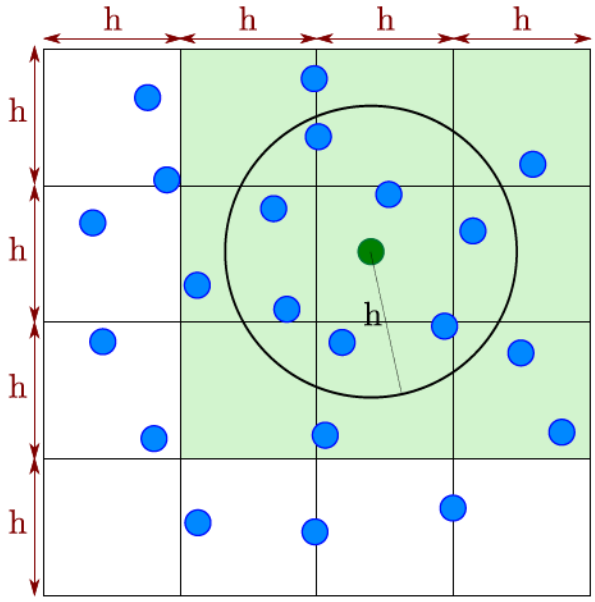


Figure 6.5 – Recherche de voisinage dans une grille régulière

Une fois cet index obtenu on peut facilement récupérer le contenu des cases adjacentes $(cx \pm 1, cy \pm 1, cz \pm 1)$.

La méthode de grille uniforme simple consiste à générer un tableau pour chaque case de la grille. Ce tableau contient l'adresse mémoire de chaque particule contenue au sein de cette case. Cependant, cette méthode est beaucoup trop coûteuse en mémoire surtout pour une programmation de type GPGPU. En effet, l'allocation dynamique est très coûteuse. À chaque pas de temps nos tableaux changent de taille en fonction du nombre de particules présentes dans chaque case.

Pour cela, nous allons adopter une approche basée sur la grille uniforme simple, mais moins contraignante du point de vue de la mémoire et surtout plus facilement parallélisable.

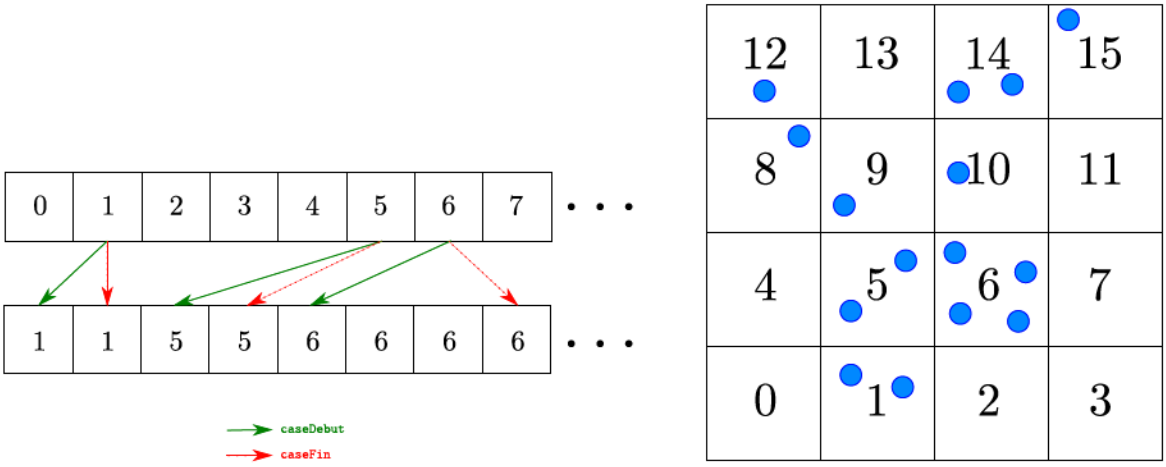
La grille triée par index

La grille triée par index basée sur l'article [Gre08] et [KS09] semble mieux répondre aux contraintes de la programmation sur le GPU. Pour cela on se sert de l'indexation linéaire cependant il nous faut utiliser d'autres tableaux. Ces tableaux décrits dans l'algorithme 6.7 vont nous permettre de facilement récupérer le voisinage.

```
struct GrilleIndexTriee
{
 int indexTri[N];
 int caseDebut[nbCasesX*nbCasesY*nbCasesZ];
 int caseFin[nbCasesX*nbCasesY*nbCasesZ];
};
```

algorithme 6.7 – Structure de la grille triée par index

`indexTri` est un tableau de taille N (nombre de particules) contenant l'ensemble des index linéaires lié aux positions des particules. Ce tableau sera trié en fonction de l'index linéaire de chaque particule, à l'aide du tri par base («radix sort» en anglais) fourni par la bibliothèque Thrust [BH11]. Selon Satsh, Harris et Garland [SHG09], le tri par base fournit d'excellents résultats sur le GPU, il est justement intégré au sein du SDK de CUDA. `caseDebut` et `caseFin` sont des tableaux permettant de stocker où commence l'index d'une case et où il se termine, visibles sur la figure 6.6a. Ils ont une taille fixée au début de la simulation `nbCasesX*nbCasesY*nbCasesZ` représente le nombre total de cases dans la simulation.



(a) Les tableaux d'index triés (b) Une grille régulière

Figure 6.6 – Représentation de l'usage d'une grille régulière et du tri par index

Le principe de l'algorithme est illustré en deux dimensions au sein de la figure 6.6. Cette illustration démontre de façon simple l'utilité de la grille triée par index. De cette manière on peut obtenir une recherche de voisinage efficace facilement parallélisable. Aucune allocation dynamique n'est nécessaire. Ainsi on obtient un gain en vitesse non négligeable. De plus, nous utilisons la mémoire partagée au sein de l'algorithme de recherche de voisinage. Nécessitant une synchronisation des threads, cette technique permet à chaque thread de connaître la valeur de l'index linéaire des cases voisines. Par exemple si un thread parcourt un élément du tableau `indexTri` d'index triés (figure 6.7), on peut facilement, à l'aide de la mémoire partagée, en déduire des threads voisins où commence cette case et où elle finit.

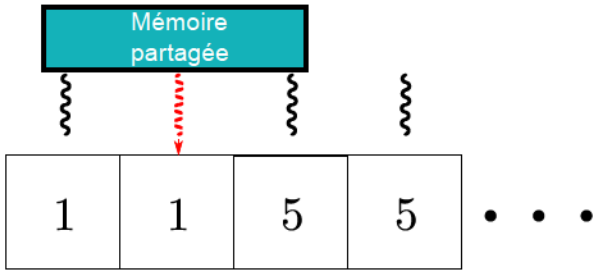


Figure 6.7 – Usage de la mémoire partagée

De cette façon, on peut optimiser la recherche pour compléter les tableaux `caseDebut` et `caseFin`. L'ensemble du code qui compose l'algorithme de grille triée par index n'est pas présent au sein de cette section.

6.3.4 Visualisation et améliorations

Visualisation

La visualisation des particules est opérée avec OpenGL. Plus précisément, nous utilisons le pipeline programmable à l'aide de l'interoperabilité OpenGL fourni par CUDA. Un «shader» (le mot est issu du verbe anglais «to shade» pris dans le sens de nuancer) est programmé pour le rendu des particules. L'interopérabilité OpenGL de CUDA nous permet de fournir directement le tableau de positions de particules au «shader» car celui-ci se situe déjà sur la mémoire de la carte graphique. Ainsi, nous évitons au maximum, les dialogues entre la mémoire du CPU et la mémoire du GPU. Le rendu de la simulation en trois dimensions est visible dans la figure 6.8.

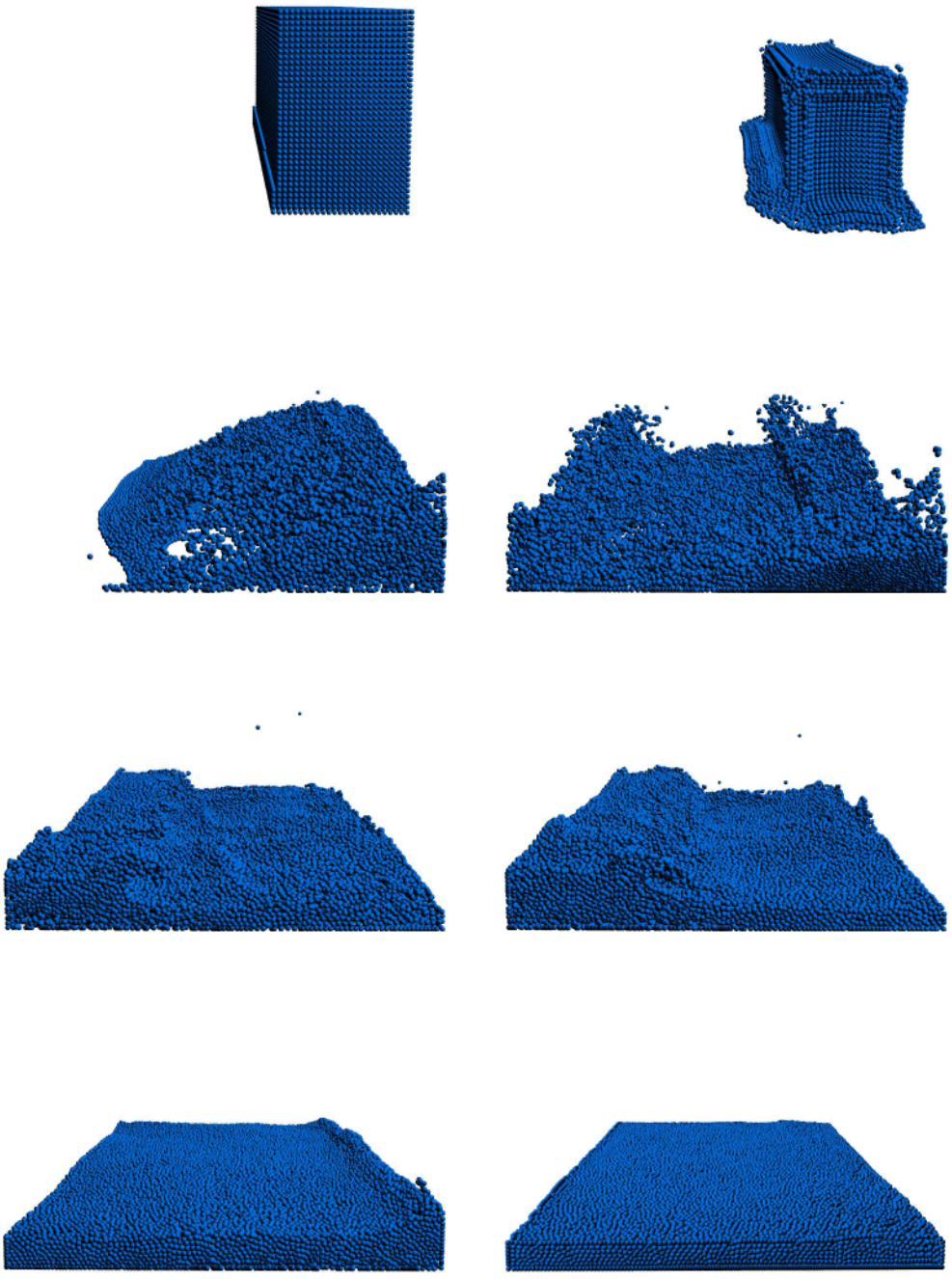


Figure 6.8 – Visualisation de la simulation de bris de barrage en trois dimensions

Améliorations

Une amélioration possible de l'algorithme serait d'utiliser l'indexation en z proposée par [GS10]. À l'aide de l'ordre en z, les particules sont triées de la même façon que pour la grille uniforme. En raison du tri, les particules qui sont dans la même cellule sont également à proximité spatiale dans la mémoire. Cela améliore la cohérence de la mémoire. De plus, les auteurs utilisent au maximum la mémoire partagée en y introduisant pour chaque bloc du GPU, le contenu d'une case. Ainsi, il est nettement plus rapide d'opérer sur une case et ses 26 cases voisines. Malheureusement, le tri de la grille dépend fortement du domaine la définissant, il peut être utile de reconsidérer notre approche en utilisant une table de hachage spatiale permettant ainsi de considérer de très grands domaines.

6.4 Performance et conclusion

6.4.1 Performance

Les tests ont été réalisés au sein de la simulation de type bris de barrage (figure 6.8). Notre configuration est un CPU Intel Core 2 Quad à 2.66 GHz avec 6GB de mémoire vive et un GPU Nvidia GeForce GTX 650 Ti avec 2GB de mémoire. Il est difficile de comparer des résultats précisément selon [WFF11] en utilisant une précision flottante. De plus le CPU et le GPU ne sont pas de mêmes générations et donc ne répondent pas aux mêmes standards. Cependant, en s'inspirant des articles publiés à SIGGRAPH [GS10, MM13], nous avons souhaité comparer, pour une simulation similaire, le temps de calcul et d'affichage sur le CPU et le GPU, tableau 6.3. Les résultats sont en images par seconde (ips) et permettent de mettre en lumière le gain de vitesse impressionnant d'une programmation orientée GPU. Nous obtenons des résultats comparables à ceux de Goswami et Schlegel [GS10] sans indexation en z :

	Nombre moyen d'images par seconde (ips)	
Nombre de particules	**GPU**	**CPU**
15 000	29 ips	1.7 ips
25 000	27 ips	1 ips
50 000	20 ips	0.4 ips
100 000	10 ips	0.2 ips
200 000	4 ips	0.1 ips

Tableau 6.3 – Performance de la simulation et du rendu pour une simulation de type bris de barrage

6.4.2 Conclusion

Dans cette section, nous avons démontré comment obtenir une simulation de particules avec rendu sur le GPU. L'usage de la carte graphique permet une accélération des calculs très importante, mais nécessite de repenser ces algorithmes et une façon de programmer plus complexe. La nécessité de connaître l'architecture matérielle est nécessaire. L'usage du GPU est très utile pour accélérer les calculs. Cependant, dans l'optique de réaliser des algorithmes préliminaires, il est utile de commencer sur une version CPU qui offre plus de maniabilité et de facilité dans une programmation de type objet. Avec l'arrivée de CUDA 6, Nvidia souhaite résoudre les manques de maniabilité. Ceci afin de permettre une utilisation plus accessible de la programmation sur GPU dans le but d'accélérer tous types de calculs (dynamiques ou non).

Conclusion

La premier chapitre de ce mémoire présente une description générale de la dynamique des fluides et du défi que représente la résolution des équations d'Euler/Navier-Stokes dans le domaine de la physique, des mathématiques et de l'infographie. Deux approches principales sont utilisées : l'approche lagrangienne et l'approche eulérienne. Chacune présente des avantages et des inconvénients. Les méthodes lagrangiennes utilisent les particules à l'aide des fonctions de lissage, ou noyaux SPH alors que les méthodes eulériennes utilisent les différences finies au sein d'une grille de calcul. Des méthodes hybrides ont été proposées et tirent partie des avantages de chaque approche. Ce mémoire se concentre essentiellement sur l'approche lagrangienne. Pour cela, le chapitre 2 permet de fournir l'ensemble des bases mathématiques des méthodes particulaires en s'appuyant sur la méthode SPH. Les chapitres 3 et 4 permettent de comprendre l'utilisation des principes mathématiques des méthodes particulaires dans un contexte de simulation de fluides. Ceci afin de résoudre les équations de Euler/Navier-Stokes. En approche lagrangienne, le fluide est traditionnellement considéré comme faiblement compressible. La raison est qu'il est nettement plus facile de calculer la pression à partir d'une équation d'état décrite dans le chapitre 3, plutôt que d'avoir à obtenir cette pression par le biais de la résolution d'une équation (à l'image des méthodes eulériennes qui utilisent une équation de Poisson). Le chapitre 4 présente les algorithmes et corrections adoptés afin de rendre le fluide incompressible. Les algorithmes performants de ces dernières années permettent un gain de performance significatif. Une revue exhaustive des dernières avancées de la simulation de fluides permet de mettre en lumière l'avantage d'utiliser une approche lagrangienne dans le but de fournir une simulation de fluide incompressible de qualité. L'application et l'explication de ces dernières avancées constituent une contribution fournissant ainsi

l'étude nécessaire pour comprendre les enjeux d'une méthode particulaire. Dans la littérature, plusieurs noyaux sont proposés afin de permettre une bonne simulation des forces qui dépendent du gradient et du Laplacien. Cela dans le but de résoudre les équations de la dynamique des fluides. Au sein du chapitre 5, nous proposons une utilisation des noyaux constants par morceaux et ce, dans un contexte lagrangien. Leur usage en une dimension [BBCE09] permet de mettre en valeur et ce en comparaison avec les méthodes SPH classiques, une grande amélioration dans les estimations des valeurs d'une fonction et de ses deux premières dérivées. Nous avons appliqué ces noyaux en deux dimensions dans une approche lagrangienne. Basée sur une résolution matricielle [CESM11], nous avons utilisé également une approche de programmation linéaire, permettant de fournir des résultats encourageants. Cependant, la simulation de fluides nécessite souvent des algorithmes très complexes et un temps de calcul important pour représenter la façon dont le fluide évolue avec l'environnement. Par conséquent, ces simulations sont généralement limitées à un rendu dit hors ligne («off line» en anglais). Cependant, avec l'augmentation continue de la puissance de calcul et de nouvelles avancées tels que les GPU, nous pouvons simuler des fluides en temps réel pour des applications interactives comme les jeux vidéo. Le chapitre 6 porte l'attention sur la programmation dite hautement parallèle. Méthodes et principes de programmation sont décrits dans le but d'obtenir un programme de rendu des fluides en trois dimensions. Ce chapitre 6 se concentre essentiellement sur les pratiques de programmation dite GPGPU («General-Purpose computation on Graphic Processing Units» en anglais) mais aussi sur la recherche de voisinages qui, dans une simulation particulaire, permet d'augmenter grandement les performances de calcul.

Annexe A

Outils mathématiques

A.1 Outils - Calcul vectoriel

A.1.1 Définitions

Au sein de ce mémoire, de nombreuses notations sont présentes. Nous allons ici définir quelques termes afin de faciliter la compréhension et l'usage de ces termes. D'une manière générale on a :

- $\vec{r}$ un vecteur représentant la position soit $\vec{r} = (x, y, z)$ dans un espace euclidien à trois dimensions ;
- $\vec{v}$ un champ vectoriel représentant la vitesse, défini par ses trois composantes : $v_x(\vec{r})$, $v_y(\vec{r})$, et $v_z(\vec{r})$;
- $\vec{a}$ un champ vectoriel représentant l'accélération, défini par ses trois composantes : $a_x(\vec{r})$, $a_y(\vec{r})$, et $a_z(\vec{r})$.

Un champ de vecteurs ou champ vectoriel est une fonction qui associe un vecteur à chaque point d'un espace euclidien. Un champ scalaire est une fonction de plusieurs variables qui associe un seul nombre réel (ou scalaire) à chaque point de l'espace.

A.1.2 Gradient

Simplement appliqué à un champ scalaire $f(x, y, z)$ en trois dimensions, l'opérateur ∇ donne le gradient du champ. Le gradient obtenu est, lui, un champ vectoriel. Le gradient représente la variation de la fonction f par rapport à la variation de ses différents paramètres. Par exemple en deux dimensions on a :

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right). \tag{A.1}$$

Et donc en trois dimensions on a :

$$\nabla f(x, y, z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right). \tag{A.2}$$

Le gradient, dans son aspect symbolique représente l'opérateur des dérivées, par exemple en trois dimensions :

$$\nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right). \tag{A.3}$$

Parfois on rencontre une notation alternative du gradient :

$$\nabla f = \frac{\partial f}{\partial \vec{x}}. \tag{A.4}$$

A.1.3 Divergence

Le produit scalaire symbolique entre l'opérateur ∇ et un champ vectoriel $\vec{v}$ donne la divergence de ce champ vectoriel. La divergence obtenue est, elle, un champ scalaire. Par exemple en deux dimensions :

$$\nabla \cdot \vec{v} = \nabla \cdot (v_x, v_y) = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}. \tag{A.5}$$

Et en trois dimensions :

$$\nabla \cdot \vec{v} = \nabla \cdot (v_x, v_y, v_z) = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}. \tag{A.6}$$

On peut facilement comprendre la notation $\nabla\cdot$, il suffit de réaliser le produit scalaire symbolique entre l'opérateur gradient et un champ vectoriel :

$$\nabla \cdot \vec{v} = \nabla \cdot (v_x, v_y, v_z) = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot (v_x, v_y, v_z), \tag{A.7}$$

$$\nabla \cdot \vec{v} = \frac{\partial}{\partial x} v_x + \frac{\partial}{\partial y} v_y + \frac{\partial}{\partial z} v_z. \tag{A.8}$$

D'une manière générale, la divergence est reliée, en physique, à l'expression locale de la propriété de conservation d'une grandeur en particulier de la densité. Cela représente le flux, par unité de volume, du vecteur $\vec{v}$ à travers une surface. Si $\nabla \cdot \vec{v} > 0$ cela signifie que nous sommes en présence d'une source. De même, si $\nabla \cdot \vec{v} < 0$, il s'agit d'un puits. Si, dans un domaine, il n'y a ni source ni puits, alors $\nabla \cdot \vec{v} = 0$ et le champ de vecteur $\vec{v}$ est dit solénoïdal ou incompressible.

A.1.4 Rotationnel

Le produit vectoriel symbolique $\times$ entre l'opérateur ∇ et un champ vectoriel $\vec{v}$ (défini par ses trois composantes : $v_x(x, y, z)$, $v_y(x, y, z)$, et $v_z(x, y, z)$ en trois dimensions), donne le rotationnel de ce champ vectoriel. Le rotationnel obtenu est, lui aussi, un champ vectoriel. Plus difficile à se représenter que le gradient ou encore la divergence, il exprime la tendance qu'ont les lignes de champ d'un champ vectoriel à tourner autour d'un point.

$$\nabla \times \vec{v} = \nabla \times (v_x, v_y, v_z) = \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z}, \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x}, \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right) \tag{A.9}$$

En deux dimensions, le rotationnel représente seulement un scalaire, la troisième composante du rotationnel en trois dimensions.

$$\nabla \times \vec{v} = \nabla \times (v_x, v_y) = \left(\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right) \tag{A.10}$$

On peut facilement comprendre la notation $\nabla\times$. Il suffit de réaliser le produit vectoriel symbolique entre l'opérateur gradient et un champ vectoriel :

$$\nabla \times \vec{v} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \times (v_x, v_y, v_z), \tag{A.11}$$

$$\nabla \times \vec{v} = \left(\frac{\partial}{\partial y}v_z - \frac{\partial}{\partial z}v_y, \frac{\partial}{\partial z}v_x - \frac{\partial}{\partial x}v_z, \frac{\partial}{\partial x}v_y - \frac{\partial}{\partial y}v_x\right). \tag{A.12}$$

A.1.5 Laplacien

La divergence du gradient d'un champ scalaire définit son Laplacien scalaire. On dit aussi que le Laplacien scalaire est l'application symbolique au champ scalaire du carré (en fait les dérivées partielles secondes) de l'opérateur ∇. Le Laplacien obtenu est lui aussi (par construction) un champ scalaire. On rencontre deux types de notation pour le Laplacien, ∇^2 ou encore la divergence du gradient $\nabla \cdot \nabla$. Soit f en deux dimensions :

$$\nabla \cdot \nabla f(x, y) = \nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{A.13}$$

Et donc en trois dimensions on a :

$$\nabla^2 f(x, y, z) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}. \tag{A.14}$$

Il est important de noter que le Laplacien peut s'appliquer à un champ vectoriel, on parle alors de Laplacien vectoriel. Le Laplacien vectoriel est tout simplement, le Laplacien scalaire appliqué à chacune des composantes $v_x(x, y, z)$, $v_y(x, y, z)$, et $v_z(x, y, z)$ du champ vectoriel $\vec{v}$:

$$\nabla^2 \vec{v} = \frac{\partial^2 v_x}{\partial x^2} + \frac{\partial^2 v_y}{\partial y^2} + \frac{\partial^2 v_z}{\partial z^2}. \tag{A.15}$$

Par définition, la formulation $\nabla^2 f = 0$ est appelée équation de Laplace. Si on décide de remplacer le 0 par une certaine quantité non nulle, $\nabla^2 f = q$ on parle alors d'équation de Poisson ou encore loi de Poisson (utilisée en statistique).

A.1.6 Identités utiles

La règle du produit :

$$\nabla(fq) = (\nabla f)\, q + f\nabla q. \tag{A.16}$$

La règle du quotient :

$$\nabla\left(\frac{f}{q}\right) = \frac{(\nabla f)\, q - f\nabla q}{q^2}, \tag{A.17}$$

$$\frac{\nabla f}{q} = \nabla\left(\frac{f}{q}\right) + \frac{f\nabla q}{q^2}. \tag{A.18}$$

La règle de la divergence :

$$\nabla \cdot (f\vec{v}) = (\nabla f) \cdot \vec{v} + f\nabla \cdot \vec{v}. \tag{A.19}$$

La règle du Laplacien obtenue en appliquant deux fois la règle du produit :

$$\nabla^2(fq) = f\left(\nabla^2 q\right) + q\left(\nabla^2 f\right) + 2\nabla f \cdot \nabla q, \tag{A.20}$$

$$q\left(\nabla^2 f\right) = \nabla^2(fq) - \left(\nabla^2 q\right) - 2\nabla f \cdot \nabla q. \tag{A.21}$$

Annexe B

Schéma d'intégration et critère du temps

B.1 Condition CFL

La condition ou critère CFL (Courant-Friedrichs-Lewy), du nom des mathématiciens Richard Courant, Kurt Friedrichs et Hans Lewy, est une condition nécessaire simple et très intuitive pour la convergence des solutions numériques, publiée en 1928 [CFL28]. Elle sert à donner le pas de discrétisation et le pas de temps sous lesquels on observe une instabilité de calcul, erreur d'approximation, qui peut grandir au fur et à mesure des calculs. Toutefois, la condition CFL est un critère nécessaire de stabilité, mais pas suffisant. Bien souvent d'autres critères entrent en jeu dans le cas de simulations complexes comme la simulation de fluide.

B.1.1 Description mathématique du critère CFL

Le cas à une dimension

$$C = \frac{v\,\Delta t}{\Delta x} \leq C_{max}, \tag{B.1}$$

Où C représente le nombre de Courant qui est une constante (sans dimension) utilisée et déterminée pour des besoins de stabilité de la simulation.

Le cas à deux dimensions et généralisation

Dans le cas à deux dimensions, on a une vitesse $\vec{v} = (v_x, v_y)$ et la condition CFL s'écrit de la manière suivante :

$$C = \frac{v_x \, \Delta t}{\Delta x} + \frac{v_y \, \Delta t}{\Delta y} \leq C_{max}. \tag{B.2}$$

Par analogie avec l'équation (B.2), on obtient le cas à n dimensions :

$$C = \Delta t \sum_{i=1}^{n} \frac{v_{x_i}}{\Delta x_i}. \tag{B.3}$$

B.1.2 Description dans le cas d'une simulation de fluide eulérienne

En utilisant une approche eulérienne, cette condition spécifie que les déplacements admissibles du fluide sur un pas de temps doivent rester petits par rapport à la taille des cellules de la grille.

Selon [BMF07], la condition CFL permet de définir le pas de temps Δt à ne pas dépasser (en considérant $\Delta x = \Delta y$) :

$$\Delta t \leq \frac{C_{max} \Delta x}{\|\vec{v}_{max}\|}. \tag{B.4}$$

Dans [FF01, FSJ01], la constante est fixée à $C_{max} = 5$. Le paramètre C_{max} n'a jamais été pris supérieur à 5 pour le schéma prédicteur-correcteur (sous-section B.2.3 ou sous-section B.2.4) ou le schéma de Runge-Kutta d'ordre 2, 3 ou 4 (sous-section B.2.5).

Remarque

Pour la méthode dite semi-lagrangienne [Sta99, Sta03], la condition CFL est automatiquement satisfaite. L'advection de la vélocité est déduite de particules virtuelles, qui, au pas de temps précédent possédaient une valeur de vélocité. Le simple fait

de remonter le flot pour en déduire une certaine caractéristique, en l'occurrence la vitesse, permet à cette méthode d'être considérée comme inconditionnellement stable.

B.1.3 Description dans le cas d'une simulation de fluide lagrangienne

Il existe une variété de méthodes et d'explications afin de définir un pas de temps en adéquation avec les solutions de type SPH. La vitesse est définie sur chaque particule, il est donc difficile de définir une vitesse limite ou plutôt un pas de temps suffisant pour conserver une stabilité. Nous allons résumer ces méthodes et en déduire une approche utile afin de respecter cette condition d'une manière lagrangienne.

Dans [GM82, Mon92, LL03], les auteurs décrivent l'approche des conditions CFL suivantes :

$$\Delta t = \underbrace{min}_{\text{i}} \left(\frac{h_i}{c} \right), \tag{B.5}$$

avec h_i qui représente le support du noyau (défini dans le chapitre 2) employé pour chaque particule i et c représente la vitesse caractéristique du milieu (souvent elle est définie comme la vitesse du son dans l'eau). En posant h_i le même pour toutes les particules, on obtient l'interprétation suivante :

$$\Delta t_f = \underbrace{min}_{\text{i}} \left(\frac{h}{\|\vec{f}_i\|} \right), \tag{B.6}$$

où $\|\vec{f}_i\|$ est la norme des forces extérieures par unité de masse (accélération en $m \cdot s^{-2}$). Cependant, dans le cas de l'usage d'un terme artificiel pour la diffusion visqueuse (viscosité) [Mon89, MFZ97], la formulation suivante de la condition CFL est utilisée :

$$\Delta t_{AV} = \underbrace{min}_{\text{i}} \left(\frac{h}{c_s + \underbrace{max}_{\text{j}} \| \frac{h \vec{v}_{ij} \vec{r}_{ij}}{\vec{r}_{ij}^2} \|} \right), \tag{B.7}$$

où c_s représente la vitesse du son dans l'eau («celerity of sound» en anglais) définie lors de l'usage de la viscosité artificielle (section 3.1.4). Dans ce cas précis on cherche

à obtenir le minimum sur chaque particule i qui possède en voisinage j particules.

En considérant l'ensemble des définitions du pas de temps précédent, on peut donc en déduire le pas de temps suivant, afin d'obtenir une simulation stable en lagrangien :

$$\Delta t = \underbrace{min}_{\text{i}} \left(\Delta t_f, \Delta t_{AV} \right). \tag{B.8}$$

Dans [Mon92, Mon94], Monaghan suggère d'utiliser une formulation similaire (B.9) mais il fixe deux constantes $\lambda_1 = 0.4$ et $\lambda_2 = 0.25$; cette définition sera appliquée et utilisée dans de nombreuses simulations comme par exemple dans [BT07] :

$$\Delta t = \underbrace{min}_{\text{i}} \left(\lambda_1 \Delta t_f, \lambda_2 \Delta t_{AV} \right). \tag{B.9}$$

Pour plus d'informations sur la définition des conditions CFL utilisées et appliquées au sein de simulations SPH, le lecteur est invité à consulter les publications suivantes : [Mon92, Mon05, LL03, LL10, MFZ97].

B.1.4 Conclusion

Dans les deux approches les conditions CFL sont des conditions plus que nécessaires. Deux approches distinctes expliquent aussi les formulations différentes. Il est à noter que l'utilisateur fixe un pas de temps et que celui-ci est subdivisé de manière adaptative pour respecter la condition de stabilité CFL. Donc pour un pas de temps donné il est possible de disposer de plusieurs sous-étapes afin d'obtenir une solution numériquement correcte.

B.2 Schéma d'intégration numérique

La résolution des équations différentielles de Navier-Stokes (ou Euler) est un point souvent abordé dans les différents travaux (articles, mémoires et thèses) relatifs à la méthode de simulation de fluide. De nombreux schémas différents sont employés mais assez peu de détails techniques sont disponibles sur l'écriture de cette résolution numérique. Ainsi, cette annexe est consacrée à cet aspect fondamental de la simulation de fluide. Toutefois, cette annexe ne contient pas l'ensemble des schémas existants mais seulement ceux utilisés dans ce mémoire.

Dans les équations suivantes, nous noterons $\vec{r}$, $\vec{v}$ et $\vec{a}$ la position, la vitesse et l'accélération d'une particule lors de l'intégration des équations différentielles du temps t au temps $t + \Delta t$, souvent appelé $t + 1$ pour des raisons de simplicité. Ces schémas peuvent également être utilisés pour intégrer la densité ρ dans l'équation de conservation de la masse. Avec $\vec{a}_t$ calculée à l'aide de l'équation de conservation du mouvement.

B.2.1 Méthode d'Euler

La méthode d'Euler est la plus basique des méthodes explicites pour l'intégration numérique des équations différentielles ordinaires. Elle a une précision de premier ordre et un coût de calcul extrêmement faible et, en tant que tel, c'est un algorithme très performant. L'algorithme prend la forme suivante :

$$\begin{aligned} \vec{v}_{t+1} &= \vec{v}_t + \Delta t\ \vec{a}_t, \\ \vec{r}_{t+1} &= \vec{r}_t + \Delta t\ \vec{v}_t. \end{aligned} \tag{B.10}$$

B.2.2 Méthode d'Euler semi-implicite

En échangeant simplement la mise à jour de la vitesse dans la mise à jour de la position on obtient une formulation implicite de méthode d'Euler. On gagne deux avantages importants : cette méthode est symplectique (éléments qui sont entrelacés les uns dans les autres) et elle fournit une plus grande stabilité que l'équation (B.10). Cela signifie que l'erreur locale (définie comme l'erreur faite en une seule étape) est

$o(\Delta t^2)$. L'erreur globale est de $o(\Delta t)$.

$$\begin{aligned} \vec{v}_{t+1} &= \vec{v}_t + \Delta t\ \vec{a}_t \\ \vec{r}_{t+1} &= \vec{r}_t + \Delta t\ \vec{v}_{t+1} \end{aligned} \tag{B.11}$$

B.2.3 Méthode de Verlet

La méthode a été développée par le physicien français Loup Verlet en 1967. L'algorithme de Verlet réduit le taux d'erreurs introduites par l'intégration en calculant la position au pas de temps suivant à partir des positions courantes et précédentes, sans faire appel à la vitesse. En utilisant deux développements de Taylor de la position $\vec{r}(t+1)$ et $\vec{r}(t-1)$.

$$\vec{r}_{t+1} = \vec{r}_t + \vec{v}_t\Delta t + \frac{1}{2}\vec{a}_t\Delta t^2 + \frac{1}{6}\frac{d^3\vec{r}_t}{dt^3}\Delta t^3 + o(\Delta t^4) \tag{B.12}$$

$$\vec{r}_{t-1} = \vec{r}_t - \vec{v}_t\Delta t + \frac{1}{2}\vec{a}_t\Delta t^2 - \frac{1}{6}\frac{d^3\vec{r}_t}{dt^3}\Delta t^3 + o(\Delta t^4) \tag{B.13}$$

En additionnant les deux expressions ensemble on obtient une nouvelle méthode d'intégration :

$$\begin{aligned} \vec{r}_{t+1} &= 2\vec{r}_t - \vec{r}_{t-1} + \Delta t^2\ \vec{a}_t, \\ \vec{v}_t &= \frac{\vec{r}_{t+1} - \vec{r}_{t-1}}{2\Delta t}. \end{aligned} \tag{B.14}$$

On remarque que la vitesse est calculée avec un certain retard. La méthode dispose d'une erreur globale de $o(\Delta t^2)$ pour la position et pour la vitesse, une erreur locale de $o(\Delta t^4)$ pour la position et de $o(\Delta t^2)$ pour la vitesse. On peut la considérer comme une méthode précise du second ordre notamment utilisée par Monaghan, [Mon05], dans un schéma plus complexe de prédiction/correction. Également dans un schéma de prédiction/relaxation utilisé par Simon Clavet [CBP05] afin de résoudre certains problèmes de stabilité.

Les inconvénients de cette méthode résident dans la nécessité de conserver les positions $\vec{r}_{t-1}$, $\vec{r}_t$ et $\vec{r}_{t+1}$, ce qui signifie également que l'algorithme ne dispose pas de point de départ.

Méthode de Verlet pour la vitesse

En modifiant quelque peu la méthode de Verlet de base, nous pouvons supprimer la faiblesse de cette méthode et disposer ainsi d'un point de départ :

$$\begin{aligned} \vec{r}_{t+1} &= \vec{r}_t + \Delta t\ \vec{v}_t + \frac{1}{2}\Delta t^2\ \vec{a}_t, \\ \vec{v}_{t+1} &= \vec{v}_t + \frac{1}{2}\left(\vec{a}_t + \vec{a}_{t+1}\right)\Delta t. \end{aligned} \tag{B.15}$$

Toutefois, l'approximation de $\vec{v}_{t+1}$ demande de connaître la valeur de $\vec{a}_{t+1}$, qui bien souvent dans le comportement d'un fluide, nécessite $\vec{v}_{t+1}$ pour être calculée. On peut donc utiliser l'approximation de $\vec{v}_{t+1}$ suivante :

$$\vec{v}_{t+1} = \frac{\vec{r}_{t+1} - \vec{r}_t}{\Delta t}. \tag{B.16}$$

B.2.4 Méthode LeapFrog ou Saute-Mouton

Leapfrog est une méthode d'intégration très similaire à la méthode de Verlet pour la vitesse. Le nom Saute-Mouton vient du fait que les positions et les vitesses sont calculées à des moments entrelacés ($t + \frac{1}{2}$ et $t - \frac{1}{2}$). La méthode est souvent utilisée dans les méthodes SPH en raison de sa bonne performance et précision, [MCG03, BT07, SP08, SP09, Mon92]. La précision de la position est la même que pour les algorithmes de Verlet, mais il est $o(\Delta t^3)$ pour la vitesse.

$$\vec{v}_{t+\frac{1}{2}} = \vec{v}_{t-\frac{1}{2}} + \Delta t\vec{a}_t \tag{B.17}$$

$$\vec{r}_{t+1} = \vec{r}_t + \Delta t\vec{v}_{t+\frac{1}{2}} \tag{B.18}$$

Avec comme condition initiale pour la vitesse, une simple intégration d'Euler :

$$\vec{v}_{-\frac{1}{2}} = \vec{v}_0 - \frac{1}{2}\Delta t\vec{a}_0. \tag{B.19}$$

Toutefois l'approximation de $\vec{v}_t$ est nécessaire pour le calcul des forces :

$$\vec{v}_t = \frac{\vec{v}_{t-\frac{1}{2}} + \vec{v}_{t+\frac{1}{2}}}{2}. \tag{B.20}$$

B.2.5 Méthode de Runge-Kutta

Les méthodes ont été nommées ainsi en l'honneur des mathématiciens Carl Runge et Martin Wilhelm Kutta lesquels élaborèrent la méthode en 1901. Ces méthodes reposent sur le principe de l'itération, c'est-à-dire qu'une première estimation de la solution est utilisée pour calculer une seconde estimation, plus précise, et ainsi de suite. On dit les méthodes car la méthode diffère selon l'ordre d'intégration. Il est important de noter que ces méthodes existent également sous leurs formulations implicites, ici nous allons seulement décrire les méthodes de type explicite.

Runge-Kutta d'ordre 1

Cette méthode est équivalente à la méthode d'Euler.

Runge-Kutta d'ordre 4 (RK4)

Cette méthode de Runge-Kutta est souvent dénommée RK4 ou simplement la méthode de Runge-Kutta. Pour la compréhension de l'algorithme on utilise :

$$\vec{a}_t = \vec{a}\left(\vec{r}_t, \vec{v}_t\right). \tag{B.21}$$

On commence en recherchant les 4 estimations suivantes :

$$\begin{aligned} \vec{r}^{\,1} &= \vec{r}_t, & \vec{r}^{\,2} &= \vec{r}_t + \frac{1}{2}\Delta t \vec{v}^{\,1}, \\ \vec{v}^{\,1} &= \vec{v}_t, & \vec{v}^{\,2} &= \vec{v}_t + \frac{1}{2}\Delta t \vec{a}^{\,1}, \\ \vec{a}^{\,1} &= \vec{a}_t, & \vec{a}^{\,2} &= \vec{a}\left(\vec{r}^{\,2}, \vec{v}^{\,2}\right), \end{aligned} \tag{B.22}$$

$$\begin{aligned}
\vec{r}^3 &= \vec{r}_t + \frac{1}{2}\Delta t\vec{v}^2, & \vec{r}^4 &= \vec{r}_t + \Delta t\vec{v}^3,\\
\vec{v}^3 &= \vec{v}_t + \frac{1}{2}\Delta t\vec{a}^2, & \vec{v}^4 &= \vec{v}_t + \Delta t\vec{a}^3,\\
\vec{a}^3 &= \vec{a}\left(\vec{r}^3, \vec{v}^3\right), & \vec{a}^4 &= \vec{a}\left(\vec{r}^4, \vec{v}^4\right).
\end{aligned} \tag{B.23}$$

Ensuite on peut facilement obtenir la nouvelle vitesse et position :

$$\begin{aligned}
\vec{v}_{t+1} &= \vec{v}_t + \frac{\Delta t}{6}\left(\vec{a}^1 + 2\vec{a}^2 + 2\vec{a}^3 + \vec{a}^4\right),\\
\vec{r}_{t+1} &= \vec{r}_t + \frac{\Delta t}{6}\left(\vec{v}^1 + 2\vec{v}^2 + 2\vec{v}^3 + \vec{v}^4\right).
\end{aligned} \tag{B.24}$$

La méthode RK4 est une méthode de quatrième ordre, ce qui signifie que l'erreur locale est de l'ordre de $o(\Delta t^5)$ ainsi que d'une erreur globale de $o(\Delta t^4)$. Cette méthode très précise est également très coûteuse en temps de calcul et donc faible en performance. En effet, il nous faut estimer quatre fois l'accélération, soit réaliser, pour un même pas de temps, quatre calculs différents.

Runge-Kutta d'ordre 2 (RK2)

La méthode RK2 est une composition de la méthode d'Euler et est très proche de la méthode de Verlet. L'erreur locale est de $o(\Delta t^3)$ et l'erreur globale est de $o(\Delta t^2)$. Elle consiste à estimer la dérivée au milieu du pas de temps et à estimer le pas de temps complet à partir de cette nouvelle estimation.
On commence par les deux estimations suivantes :

$$\begin{aligned}
\vec{r}^1 &= \vec{r}_t, & \vec{r}^2 &= \vec{r}_t + \frac{1}{2}\Delta t\vec{v}^1,\\
\vec{v}^1 &= \vec{v}_t, & \vec{v}^2 &= \vec{v}_t + \frac{1}{2}\Delta t\vec{a}^1,\\
\vec{a}^1 &= \vec{a}_t, & \vec{a}^2 &= \vec{a}\left(\vec{r}^2, \vec{v}^2\right).
\end{aligned} \tag{B.25}$$

Puis on en déduit la position et la vitesse au pas de temps complet :

$$\begin{aligned}
\vec{v}_{t+1} &= \vec{v}_t + \Delta t\vec{a}^2,\\
\vec{r}_{t+1} &= \vec{r}_t + \Delta t\vec{v}^2.
\end{aligned} \tag{B.26}$$

On reconnaît ainsi que la méthode d'intégration utilisée pour les temps intermédiaires est celle du point milieu. Souvent appelée «Midpoint method» en anglais. La méthode du point milieu est une méthode Runge-Kutta du second ordre. Méthode très utilisée notamment au sein des méthodes eulériennes. Nous l'avons utilisée dans le cadre d'une simulation de fluide de type PIC/FLIP (hybride) basée sur les écrits de Brackbill, [BKR88] et [BR86].

B.2.6 Conclusion

Cette section décrit un ensemble de méthodes d'intégrations numériques. L'intégration est un des problèmes souvent rencontrés en simulation de fluide. C'est pour cela que les intégrations d'équations différentielles représentent un vaste champ d'étude, que ce soit en mathématiques pures ou en mathématiques appliquées. Le tableau B.1 décrit et compare l'erreur globale observée pour une équation différentielle de type $\frac{dx(t)}{dt} = -2x(t)$ où la solution analytique est $x(t) = e^{-2t}$ et nous permet donc de conclure que les méthodes Runge-Kutta sont des méthodes très précises (les meilleures) mais aux dépens de l'aspect temps réel (gain de vitesse). Ces méthodes sont plus coûteuses en temps de calcul.

Méthodes	**L'erreur globale**
Méthode d'Euler	0.028
Méthode du point milieu (RK2)	-0.00211
Méthode Runge-Kutta d'ordre 3 (RK3)	0.000105
Méthode Runge-Kutta d'ordre 4 (RK4)	-4.2651×10^{-6}
Méthode d'Euler semi-implicite	-0.0261
Méthode du point milieu implicite	0.000904

Tableau B.1 – Tableau représentant l'erreur globale à $t = 1$ avec $\Delta t = 0.1$

Bibliographie

[ADO92] E. Anderson, J. Dongarra, et S. Ostrouchov, *Lapack working note 41 : Installation guide for lapack.* University of Tennessee, Computer Science Department, 1992.

[And95] J. Anderson, *Computational Fluid Dynamics : The Basics with Applications.*, 1995.

[Bat00] G. Batchelor, *An introduction to fluid dynamics.* Cambridge : Cambridge University Press, 2000.

[BBCE09] J.-M. Belley, P. Belley, F. Colin, et R. Egli, « Non-smooth kernels for meshfree methods in fluid dynamics, » *Computers & Mathematics with Applications*, vol. 58, no. 6, pp. 1253–1272, septembre 2009.

[BCL99] H. Brezis, P. Ciarlet, et J. Lions, *Analyse fonctionnelle : Théorie et applications*, série Collection Mathématiques appliquées pour la maîtrise. Dunod, 1999.

[BH11] N. Bell et J. Hoberock, « Thrust : A productivity-oriented library for CUDA, » pp. 359–373, 2011. Disponible à http://thrust.github.io/

[BKR88] J. Brackbill, D. Kothe, et H. Ruppel, « FLIP : A low-dissipation, particle-in-cell method for fluid flow, » *Computer Physics Communications*, vol. 48, pp. 25–38, 1988.

[BMF07] R. Bridson et M. Müller-Fischer, « FLUID SIMULATION SIGGRAPH 2007 Course Notes, » dans *ACM SIGGRAPH 2007 Courses*, 2007, pp. 1–81.

[BR86] J. Brackbill et H. Ruppel, « FLIP : A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions, » *Journal of Computational Physics*, 1986.

[BT07] M. Becker et M. Teschner, « Weakly compressible SPH for free surface flows, » *Proceedings of the 2007 ACM SIGGRAPH*, vol. 1, pp. 209–217, 2007.

[CBP05] S. Clavet, P. Beaudoin, et P. Poulin, « Particle-based viscoelastic fluid simulation, » *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, p. 219, 2005.

[CEL06] F. Colin, R. Egli, et F. Lin, « Computing a null divergence velocity field using smoothed particle hydrodynamics, » *Journal of Computational Physics*, no. November, pp. 1–28, 2006.

[CESM11] F. Colin, R. Egli, et A. Serghini Mounim, « A stabilized meshfree reproducing kernel-based method for convection–diffusion problems, » *International Journal for Numerical Methods in Engineering*, vol. 87, no. 9, pp. 869–888, 2011.

[CFL28] R. Courant, K. Friedrichs, et H. Lewy, « Über die partiellen Differenzengleichungen der mathematischen Physik, » *Mathematische Annalen*, vol. 100, no. 1, pp. 32–74, décembre 1928.

[CMM+10] S. Carlson, G. Margulis, R. Melrose, Y. Siu, et A. Wiles, « First Clay Mathematics Institute Millennium Prize Announced Today Prize for Resolution of the Poincaré Conjecture a Awarded to Dr. Grigoriy Perelman, » pp. 1–8, 2010.

[CSB09] V. Couvert, S. Steer, et M. Baudin, « Optimization in Scilab, » Technical report, Scilab Consortium., Rapport technique March, 2009. Disponible à http://forge.scilab.org/index.php/p/docoptimscilab/

[CT05] E. Candes et T. Tao, « Decoding by linear programming, » *IEEE Transactions on Information Theory*, vol. 40698, no. December, pp. 1–22, 2005.

[DG96] M. Desbrun et M.-P. Gascuel, « Smoothed Particles : A new paradigm for animating highly deformable bodies, » dans *Computer Animation and*

Simulation '96, G. Boulic, Ronan and Hégron, éditeur. Springer Vienna, 1996, pp. 61–76.

[Fef00] C. Fefferman, « Existence and smoothness of the Navier-Stokes equation, » *The millennium prize problems*, no. 1, pp. 1–6, 2000.

[FF01] N. Foster et R. Fedkiw, « Practical animation of liquids, » *Proceedings of the 28th annual conference on computer graphics and interactive techniques*, pp. 12–17, 2001.

[FSJ01] R. Fedkiw, J. Stam, et H. Jensen, « Visual simulation of smoke, » *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 15–22, 2001.

[GH11] C. Gregg et K. Hazelwood, « Where is the data ? Why you cannot debate CPU vs. GPU performance without the answer, » *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium*, pp. 134–144, 2011.

[GM77] R. Gingold et J. Monaghan, « smoothed particle hydrodynamics-theory and application to non-spherical stars, » *Monthly notices of the royal astronomical society*, 1977.

[GM82] R. Gingold et J. Monaghan, « Kernel estimates as a basis for general particle methods in hydrodynamics, » *Journal of Computational Physics*, vol. 46, no. 3, pp. 429–453, 1982.

[Gre08] S. Green, « Particle Simulation using CUDA, » Rapport technique, juillet 2008.

[GS10] P. Goswami et P. Schlegel, « Interactive SPH simulation and rendering on the GPU, » *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 55–64, 2010.

[Hwu14] W.-m. W. Hwu, « Heterogeneous Parallel Programming, » 2014. Disponible à https://www.coursera.org/course/hetero

[KS09] J. Kalojanov et P. Slusallek, « A parallel algorithm for construction of uniform grids, » *Proceedings of the Conference on High Performance Graphics*, pp. 1–6, 2009.

[KWm12] D. Kirk et W. Wen-mei, *Programming Massively Parallel Processors : A Hands-on Approach*. Newnes, 2012.

[LL03] M. Liu et G. Liu, *Smoothed particle hydrodynamics : a meshfree particle method*. Newnes, 2003.

[LL10] M. Liu et G. Liu, « Smoothed particle hydrodynamics (SPH) : an overview and recent developments, » *Archives of computational methods in engineering*, pp. 25–76, 2010.

[LSSF06] F. Losasso, T. Shinar, A. Selle, et R. Fedkiw, « Multiple interacting liquids, » *ACM Transactions on Graphics ...*, vol. 1, no. 212, pp. 812–819, 2006.

[Luc77] L. Lucy, « A numerical approach to the testing of the fission hypothesis, » *The astronomical journal*, vol. 82, pp. 1013–1024, 1977.

[Mak00] A. O. Makhorin, « GLPK (GNU Linear Programming Kit), » 2000. Disponible à https://www.gnu.org/software/glpk/

[MCG03] M. Müller, D. Charypar, et M. Gross, « Particle-based fluid simulation for interactive applications, » *Proceedings of the 2003 ACM*, pp. 154–160, 2003.

[MFZ97] J. P. Morris, P. J. Fox, et Y. Zhu, « Modeling Low Reynolds Number Incompressible Flows Using SPH, » *Journal of Computational Physics*, vol. 136, no. 1, pp. 214–226, septembre 1997.

[MG83] J. Monaghan et R. Gingold, « Shock simulation by the particle method *SPH*, » *Journal of Computational Physics*, vol. 52, pp. 374–389, 1983.

[MHHR07] M. Müller, B. Heidelberger, M. Hennix, et J. Ratcliff, « Position based dynamics, » *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 109–118, avril 2007.

[Mis03] B. Mishra, « A review of computer simulation of tumbling mills by the discrete element method : Part I—contact mechanics, » *International Journal of Mineral Processing*, vol. 71, no. 1-4, pp. 73–93, septembre 2003.

[MK99] J. Monaghan et A. Kos, « Solitary Waves on a Cretan Beach, » *Journal of Waterway, Port, Coastal, and Ocean Engineering*, vol. 125, no. 3, pp. 145–155, 1999.

[MM13] M. Macklin et M. Müller, « Position Based Fluids, » *ACM Transactions on Graphics*, vol. 32, no. 4, pp. 1–5, 2013.

[Mon85] J. Monaghan, « Particle methods for hydrodynamics, » *Computer Physics Reports*, vol. 3, pp. 71–124, 1985.

[Mon89] J. Monaghan, « On the problem of penetration in particle methods, » *Journal of Computational physics*, vol. 89, no. 1, pp. 1–15, 1989.

[Mon92] J. Monaghan, « Smoothed particle hydrodynamics, » *Annual review of astronomy and astrophysics*, vol. 30, no. 1, pp. 543–574, janvier 1992.

[Mon94] J. Monaghan, « Simulating free surface flows with SPH, » *Journal of computational physics*, vol. 110, no. 2, pp. 399–406, 1994.

[Mon00] J. Monaghan, « SPH without a Tensile Instability, » *Journal of Computational Physics*, vol. 159, no. 2, pp. 290–311, avril 2000.

[Mon05] J. Monaghan, « Smoothed particle hydrodynamics, » *Reports on Progress in Physics*, vol. 68, no. 8, pp. 1703–1759, août 2005.

[NHP07] L. Nyland, M. Harris, et J. Prins, « Fast N-Body Simulation with CUDA, » dans *GPU gems 3*, 2007, no. 31, ch. 31, pp. 1–9.

[Nvi14] Nvidia, *CUDA C Programming guide*, 2014. Disponible à http://docs.nvidia.com/cuda/

[Sam14] E. Samson, « SIMULATION DE FLUIDE AVEC DES NOYAUX CONSTANTS PAR MORCEAUX, » Mémoire de maîtrise, Département d'informatique, Université de Sherbooke, 2014.

[SB12] H. Schechter et R. Bridson, « Ghost SPH for animating water, » *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 1–8, juillet 2012.

[She68] D. Shepard, « A two-dimensional interpolation function for irregularly-spaced data, » *Proceedings of the 1968 23rd ACM National Conference*, pp. 517–524, 1968.

[SHG09] N. Satish, M. Harris, et M. Garland, « Designing efficient sorting algorithms for manycore GPUs, » *2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–10, mai 2009.

[SP08] B. Solenthaler et R. Pajarola, « Density contrast SPH interfaces, » *Proceedings of the 2008 ACM SIGGRAPH*, pp. 211–218, 2008.

[SP09] B. Solenthaler et R. Pajarola, « Predictive-corrective incompressible SPH, » *ACM Transactions on Graphics*, vol. 28, no. 3, juillet 2009.

[SSC+13] A. Stomakhin, C. Schroeder, L. Chai, J. Teran, et A. Selle, « A material point method for snow simulation, » *ACM Transactions on Graphics*, vol. 32, no. 4, p. 1, juillet 2013.

[Sta99] J. Stam, « Stable fluids, » *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128, 1999.

[Sta03] J. Stam, « Real-time fluid dynamics for games, » *Proceedings of the game developer conference*, vol. 18, p. 25, 2003.

[WFF11] N. Whitehead et A. Fit-Florea, « Precision & performance : Floating point and IEEE 754 compliance for NVIDIA GPUs, » *rn (A+ B)*, vol. 21, pp. 1–18, 2011.

[YT13] J. Yu et G. Turk, « Reconstructing surfaces of particle-based fluids using anisotropic kernels, » *ACM Transactions on Graphics*, vol. 32, no. 1, pp. 1–12, janvier 2013.